User Manual
Nexto Series CPU
NX3030

MU214615 Rev. F

September 25, 2024

www.altus.com.br

According to current legislation in Brazil, the Consumer Defense Code, we are giving the following information to clients who use our products, regarding personal safety and premises.

The industrial automation equipment, manufactured by Altus, is strong and reliable due to the stringent quality control it is subjected to. However, any electronic industrial control equipment (programmable controllers, numerical commands, etc.) can damage machines or processes controlled by them when there are defective components and/or when a programming or installation error occurs. This can even put human lives at risk. The user should consider the possible consequences of the defects and should provide additional external installations for safety reasons. This concern is higher when in initial commissioning and testing.

The equipment manufactured by Altus does not directly expose the environment to hazards, since they do not issue any kind of pollutant during their use. However, concerning the disposal of equipment, it is important to point out that built-in electronics may contain materials which are harmful to nature when improperly discarded. Therefore, it is recommended that whenever discarding this type of product, it should be forwarded to recycling plants, which guarantee proper waste management.

It is essential to read and understand the product documentation, such as manuals and technical characteristics before its installation or use. The examples and figures presented in this document are solely for illustrative purposes. Due to possible upgrades and improvements that the products may present, Altus assumes no responsibility for the use of these examples and figures in real applications. They should only be used to assist user trainings and improve experience with the products and their features.

Altus warrants its equipment as described in General Conditions of Supply, attached to the commercial proposals.

Altus guarantees that their equipment works in accordance with the clear instructions contained in their manuals and/or technical characteristics, not guaranteeing the success of any particular type of application of the equipment.

Altus does not acknowledge any other guarantee, directly or implied, mainly when end customers are dealing with third-party suppliers. The requests for additional information about the supply, equipment features and/or any other Altus services must be made in writing form. Altus is not responsible for supplying information about its equipment without formal request. These products can use EtherCAT® technology (www.ethercat.org).

COPYRIGHTS

OPEN SOURCE SOFTWARE NOTICE

To obtain the source code under GPL, LGPL, MPL and other open source licenses, that is contained in this product, please contact opensource@altus.com.br. In addition to the source code, all referred license terms, warranty disclaimers and copyright notices may be disclosed under request.

Contents

1. Introduction

Nexto Series programmable controllers are the ultimate solution for industrial automation and system control. With high technology embedded, the products of the family are able to control, in a distributed and redundant way, complex industrial systems, machines, high performance production lines and the most advanced processes of Industry 4.0. Modern and high-speed, the Nexto series uses cutting-edge technology to provide reliability and connectivity, helping to increase the productivity of different businesses.

Compact, robust and with high availability, the series products have excellent processing performance and rack expansion possibilities. Its architecture allows easy integration with supervision, control and field networks, in addition to PLC redundancy. The series equipment also offers advanced diagnostics and hot swapping, minimizing or eliminating maintenance downtime and ensuring a continuous production process.

Figure 1: NX3030

1.1. Nexto Series

Nexto Series is a powerful and complete series of Programmable Controllers (PLC) with exclusive and innovative characteristics. Due to its flexibility, functional design, advanced diagnostic resources and modular architecture, the Nexto PLC can be used to control systems in small, medium and large scale applications.

Nexto Series architecture has a great variety of input and output modules. These modules combined with a powerful processor and a high speed bus based on Ethernet, fit to several application kinds as high speed control for small machines, complex distributed processes, redundant applications and systems with a great number of I/O as building automation. Furthermore, Nexto Series has modules for motion control, communication modules encompassing the most popular field networks among other features.

Nexto Series uses an advanced technology in its bus, which is based on a high speed Ethernet interface, allowing input and output information and data to be shared between several controllers inside the same system. The system can be easily divided and distributed throughout the whole field, allowing the use of bus expansion with the same performance of a local module, turning possible the use of every module in the local frame or in the expansion frames with no restrictions. For interconnection between frames expansions a simple standard Ethernet cable is used.

Figure 2: Nexto Series – Overview

1.2. Innovative Features

Nexto Series brings to the user many innovations regarding utilization, supervision and system maintenance. These features were developed focusing a new concept in industrial automation.

Battery Free Operation: Nexto Series does not require any kind of battery for memory maintenance and real time clock operation. This feature is extremely important because it reduces the system maintenance needs and allows the use in remote locations where maintenance can be difficult to be performed. Besides, this feature is environmentally friendly.

Easy Plug System: Nexto Series has an exclusive method to plug and unplug I/O terminal blocks. The terminal blocks can be easily removed with a single movement and with no special tools. In order to plug the terminal block back to the module, the frontal cover assists the installation procedure, fitting the terminal block to the module.

Multiple Block Storage: Several kinds of memories are available to the user in Nexto Series CPUs, offering the best option for any user needs. These memories are divided in volatile memories and non-volatile memories. For volatile memories, Nexto Series CPUs offer addressable input (%I), addressable output (%Q), addressable memory (%M), data memory and redundant data memory. For applications that require non-volatile functionality, Nexto Series CPUs bring retain addressable memory (%Q), retain data memory, persistent addressable memory (%Q), persistent data memory, program memory, source code memory, CPU file system (doc, PDF, data) and memory card interface.

One Touch Diag: One Touch Diag is an exclusive feature that Nexto Series brings to PLCs. With this new concept, the user can check diagnostic information of any module present in the system directly on CPU's graphic display with one single press in the diagnostic switch of the respective module. OTD is a powerful diagnostic tool that can be used offline (without supervisor or programmer), reducing maintenance and commissioning times.

OFD – On Board Full Documentation: Nexto Series CPUs are capable of storing the complete project documentation in its own memory. This feature can be very convenient for backup purposes and maintenance, since the complete information is stored in a single and reliable place.

ETD – Electronic Tag on Display: Another exclusive feature that Nexto Series brings to PLCs is the Electronic Tag on Display. This new functionality brings the process of checking the tag names of any I/O pin or module used in the system directly to the CPU's graphic display. Along with this information, the user can check the description, as well. This feature is extremely useful during maintenance and troubleshooting procedures.

DHW – Double Hardware Width: Nexto Series modules were designed to save space in user cabinets or machines. For this reason, Nexto Series delivers two different module widths: Double Width (two backplane rack slots are required) and Single Width (only one backplane rack slot is required). This concept allows the use of compact I/O modules with a high-density of I/O points along with complex modules, like CPUs, fieldbus masters and power supply modules.

High-speed CPU: All Nexto Series CPUs were designed to provide an outstanding performance to the user, allowing the coverage of a large range of applications requirements.

iF Product Design Award 2012: Nexto Series was the winner of iF Product Design Award 2012 in industry + skilled trades group. This award is recognized internationally as a seal of quality and excellence, considered the Oscars of the design in Europe..

1.3. Documents Related to this Manual

In order to obtain additional information regarding the Nexto Series, other documents (manuals and technical features) besides this one, may be accessed. These documents are available in its last version on the site https://www.altus.com.br/en/.

Each product has a document designed by Technical Features (CE), where the product features are described. Furthermore, the product may have Utilization Manuals (the manuals codes are listed in the CE).

For instance, the NX2020 module has the information for utilization features and purchasing on its CE. On another hand, the NX5001 has, besides the CE, a User Manual (MU).

It is advised the following documents as additional information source:

Code	Description	Language
CE114000	Nexto Series – Technical Characteristics	English
CT114000	Série Nexto – Características Técnicas	Portuguese
CE114102	NX3030 Technical Characteristics	English
CT114102	Características Técnicas NX3030	Portuguese
CE114200	NX8000 Power Supply Module Technical Characteristics	English
CT114200	Características Técnicas Fonte de Alimentação NX8000	Portuguese
CE114700	Nexto Series Backplane Racks Technical Characteristic	English
CT114700	Características Técnicas dos Bastidores da Série Nexto	Portuguese
CE114810	Nexto Series Accessories for Backplane Rack Technical Characteristics	English
CT114810	Características Técnicas Acessórios para Bastidor Série Nexto	Portuguese
CE114900	NX4010 Redundancy Link Module Technical Characteristics	English
CT114900	Características Técnicas do Módulo de Redundância NX4010	Portuguese
CE114902	Nexto Series PROFIBUS-DP Master Technical Characteristics	English
CT114902	Características Técnicas do Mestre PROFIBUS-DP da Série Nexto	Portuguese
CE114903	Nexto Series Ethernet Module Technical Characteristics	English
CT114903	Características Técnicas Módulo Ethernet Série Nexto	Portuguese
CE114908	NX5110 and NX5210 PROFIBUS-DP Heads Technical Characteristics	English
CT114908	Características Técnicas Interfaces Cabeça PROFIBUSDP NX5110 e NX5210	Portuguese
CS114908	Especificaciones y Configuraciones PROFIBUS-DP Interfaz Cabezas NX5110 y NX5210	Spanish
CT112500	Características Técnicas do Painel de Controle de Redundância PX2612	Português
MU214600	Nexto Series User Manual	English
MU214000	Manual de Utilização Série Nexto	Portuguese
MU214615	NX3030 CPU User Manual	English
MU214103	Manual de Utilização UCP NX3030	Portuguese
MU299609	MasterTool IEC XE User Manual	English
MU299048	Manual de Utilização MasterTool IEC XE	Portuguese

Code	Description	Language
MP399609	MasterTool IEC XE Programming Manual	English
MP399048	Manual de Programação MasterTool IEC XE	Portuguese
MU214601	NX5001 PROFIBUS DP Master User Manual	English
MU214001	Manual de Utilização Mestre PROFIBUS-DP NX5001	Portuguese
MU214608	Nexto PROFIBUS-DP Head Utilization Manual	English
MU214108	Manual de Utilização da Cabeça PROFIBUS-DP Nexto	Portuguese
MU219000	Ponto Series Utilization Manual	English
MU209000	Manual de Utilização da Série Ponto	Portuguese
MU209508	Manual de Utilização Cabeça PROFIBUS PO5063V1 e Cabeça Redundante PROFIBUS PO5063V5	Portuguese
MU219511	PO5064 PROFIBUS Head and PO5065 Redundant PROFIBUS Head Utilization Manual	English
MU209511	Manual de Utilização Cabeça PROFIBUS PO5064 e Cabeça Redundante PROFIBUS PO5065	Portuguese
MU209020	Manual de Utilização Rede HART sobre PROFIBUS	Portuguese
MU223603	IEC 60870-5-104 Server Device Profile Document	English
MU214603	Nexto Series HART Manual	English
MU214606	MQTT User Manual	English
MU214609	OPC UA Server for Altus Controllers User Manual	English
MU214610	Advanced Control Functions User Manual	English
MU214621	Nexto Series PROFINET Manual	English
NAP151	Utilização do Tunneller OPC	Portuguese

Table 1: Related Documents

1.4. Visual Inspection

Before resuming the installation process, it is advised to carefully visually inspect the equipment, verifying the existence of transport damage. Verify if all parts requested are in perfect shape. In case of damages, inform the transport company or Altus distributor closest to you.

CAUTION

Before taking the modules off the case, it is important to discharge any possible static energy accumulated in the body. For that, touch (with bare hands) on any metallic grounded surface before handling the modules. Such procedure guaranties that the module static energy limits are not exceeded.

It's important to register each received equipment serial number, as well as software revisions, in case they exist. This information is necessary, in case the Altus Technical Support is contacted.

1.5. Technical Support

For Altus Technical Support contact in São Leopoldo, RS, call +55 51 3589-9500. For further information regarding the Altus Technical Support existent on other places, see https://www.altus.com.br/en/ or send an email to altus@altus.com.br.

If the equipment is already installed, you must have the following information at the moment of support requesting:

- The model from the used equipments and the installed system configuration
- The product serial number
- The equipment revision and the executive software version, written on the tag fixed on the product's side
- CPU operation mode information, acquired through MasterTool IEC XE
- The application software content, acquired through MasterTool IEC XE
- Used programmer version

1.6. Warning Messages Used in this Manual

In this manual, the warning messages will be presented in the following formats and meanings:

DANGER

Reports potential hazard that, if not detected, may be harmful to people, materials, environment and production.

CAUTION

Reports configuration, application or installation details that must be taken into consideration to avoid any instance that may cause system failure and consequent impact.

ATTENTION

Identifies configuration, application and installation details aimed at achieving maximum operational performance of the system.

2. Technical Description

This chapter presents all technical features from NX3030.

2.1. Panels and Connections

The following figure shows the CPU front panel.

Figure 3: NX3030

As it can be seen on the figure, on the front panel upper part is placed the graphic display used to show the whole system status and diagnostics, including the specific diagnostics of each module. The graphic display also offers an easy-to-use menu which brings to the user a quick mode for parameters reading or defining, such as: inner temperature (reading only) and local time (reading only).

Just below the graphic display, there are 2 LEDs used to indicate alarm diagnostics and watchdog circuit. The table below shows the LEDs description. For further information regarding the LEDs status and meaning, see Diagnostics via LED section.

LED	Description
DG	Diagnostics LED
WD	Watchdog LED

Table 2: LEDs Description

Nexto Series CPUs has two switches available to the user. The table below shows the description of these switches. For further information regarding the diagnostics switch, see sections One Touch Diag and CPU's Informative and Configuration Menu. For further information regarding the MS switch, see section Memory Card.

Keys	Description
Diagnostics Switch	Switch placed on the module upper part. Used for diagnostics visualization on the graphic display or for navigation through the informative menu and CPU configuration.
MS	Switch placed on the frontal panel. Used to securely remove the memory card.

Table 3: Keys Description

On the frontal panel the connection interfaces of Nexto Series CPUs are available. The table below presents a brief description of these interfaces.

Interfaces	Description
NET 1	RJ45 communication connector 10/100Base-TX standard. Allows the point to point or network communication. For further utilization information, see Ethernet Interfaces Configuration section.
NET 2	RJ45 communication connector 10/100Base-TX standard. Allows the point to point or network. For further utilization information, see Ethernet Interfaces Configuration section.
COM 1	DB9 female connector for RS-232 communication standard. Allows the point to point or network. For further utilization information, see Serial Interfaces Configuration section.
COM 2	For further utilization information, see Serial Interfaces Configuration section.
MEMORY SLOT	Memory card slot. Allows the use of a memory card for different types of data storage such as: user logs, project documentation and files. For further utilization information, see Memory Card section.

Table 4: Connection Interfaces

2.2. General Features

2.2.1. Common General Features

	NX3030
Backplane rack occupation	2 sequential slots
Power supply integrated	No
Ethernet TCP/IP local interface	2
Serial Interface	2
CAN Interface	No
USB Port Host	No
Memory Card Interface	Yes
Real time clock (RTC)	Yes Resolution of 1 ms and maximum variance of 2 s per day.
Watchdog	Yes

	NX3030
Status and diagnostic Indication	Graphic display LEDs System Web Page CPU internal memory
Programming languages	Structured Text (ST) Ladder Diagram (LD) Sequential Function Chart (SFC) Function Block Diagram (FBD) Continuous Function Chart (CFC)
Tasks	Cyclic (periodic) Event (software interruption) External (hardware interruption) Freewheeling (continuous) Status (software interruption)
Online changes	Yes
Maximum number of tasks	32
Maximum number of expansion bus	24
Bus expansion redundancy support	Yes
Maximum number of I/O modules on the bus	128
Maximum number of additional Ethernet TCP/IP interface modules	6
Ethernet TCP/IP interface redundancy support	Yes
Maximum number of PROFIBUS-DP network (using master modules PROFIBUS-DP)	4
PROFIBUS-DP network redundancy support	Yes
Redundancy support (half-clusters)	Yes
Hot Swap support	Yes
Event oriented data reporting (SOE) **Protocol** **Maximum Event Queue Size**	Yes DNP3 1000
Web pages development (available through the HTTP protocol)	No
One Touch Diag (OTD)	Yes
Electronic Tag on Display (ETD)	Yes

Table 5: Common Features

Notes:

Real Time Clock (RTC): The retention time, time that the real time clock will continue to update the date and time after a CPU power down, is 15 days for operation at 25 °C. At the maximum product temperature, the retention time is reduced to 10 days.

Maximum number of I/O modules on bus: The maximum number of I/O modules refers to the sum of all modules on the local bus and expansions.

Event Log (SOE): Data types are found in the DNP3 Device Profile.

2.2.2. Standards and Certifications

Standards and Certifications	
IEC	61131-2: Industrial-process measurement and control - Programmable controllers - Part 2: Equipment requirements and tests 61131-3: Programmable controllers - Part 3: Programming languages
	DNV Type Approval – DNV-CG-0339 (TAA000013D)
	2014/30/EU (EMC) 2014/35/EU (LVD) 2011/65/EU and 2015/863/EU (ROHS)
	S.I. 2016 No. 1091 (EMC) S.I. 2016 No. 1101 (Safety) S.I. 2012 No. 3032 (ROHS)
	UL/cUL Listed – UL 61010-1 UL 61010-2-201 (file E473496)
	TR 004/2011 (LVD) CU TR 020/2011 (EMC)

Table 6: Standards and Certifications

2.2.3. Memory

	NX3030
Addressable input variables memory (%I)	96 Kbytes
Addressable output variables memory (%Q)	96 Kbytes
Direct representation variable memory (%M)	64 Kbytes
Symbolic variable memory	6 Mbytes
Persistent or Retain symbolic variables memory	112 Kbytes
Full Redundant Data Memory	736 Kbytes
Direct representation input variable memory (%I)	80 Kbyte
Direct representation output variable memory (%Q)	80 Kbytes
Direct representation variable memory (%M)	64 Kbytes
Symbolic variable memory	512 Kbytes
Program memory	8 Mbytes
Source code memory (backup)	120 Mbytes
User files memory	32 Mbytes

Table 7: Memory

Notes:

Addressable input variables memory (%I): Area where the addressable input variables are stored. Addressable variables means that the variables can be accessed directly using the desired address. For instance: %IB0, %IW100. Addressable input variables can be used for mapping digital or analog input points. As reference, 8 digital inputs can be represented per byte and one analog input point can be represented per two bytes.

Total addressable output variables memory (%Q): Area where the addressable output variables are stored. Addressable variables means that the variables can be accessed directly using the desired address. For instance: %QB0, %QW100. Addressable output variables can be used for mapping digital or analog output points. As reference, 8 digital outputs can be represented per byte and one analog output point can be represented per two bytes. The addressable output variables can be configured as retain, persistent or redundant variables, but the total size is not modified due to configuration.

Nexto Series NX3030 CPU allows the definition of an area of redundant variables into the %Q direct representation output variables memory area. The subset of memory types of output direct representation variables are part of the total available memory.

Addressable variables memory (%M): Area where the addressable marker variables are stored. Addressable variables means that the variables can be accessed directly using the desired address. For instance: %MB0, %MW100.

Symbolic variables memory: Area where the symbolic variables are allocated. Symbolic variables are IEC variables created in POUs and GVLs during application development, which are not addressed directly in memory. Symbolic variables can be defined as retentive or persistent, in which case the memory areas of retentive symbolic variables or memory of persistent symbolic variables respectively will be used. The PLC system allocates variables in this area, so the space available for the allocation of variables created by the user is lower than that reported in the table. The occupation of the system variables depends on the characteristics of the project (number of modules, drivers, etc...), so it is recommended to observe the space available in the compilation messages of the MasterTool IEC XE tool.

Persistent or Retain symbolic variables memory: Area where are allocated the retentive symbolic variables. The retentive data keep its respective values even after a CPU's cycle of power down and power up. The persistent data keep its respective values even after the download of a new application in the CPU.

ATTENTION

The declaration and use of symbolic persistent variables should be performed exclusively through the *Persistent Vars* object, which may be included in the project through the tree view in *Application -> Add Object -> Persistent Variables*. It should not be used to *VAR PERSISTENT* expression in the declaration of field variables of POUs.

The full list of when the symbolic persistent variables keep their values and when the value is lost can be found in the table below. Besides the persistent area size declared in the table above, are reserved these 44 bytes to store information about the persistent variables (not available for use).

The table below shows the behavior of retentive and persistent variables for different situations in which "-" means the value is lost and "X" means the value is kept.

Command/Operation	VAR	VAR RETAIN	VAR PERSISTENT
Power cycle	-	X	X
Reset warm	-	X	X
Reset cold	-	-	X
Reset origin	-	-	-
Remove CPU with integrated power supply from the rack while powered on	-	X	X
Remove the power supply or a CPU without integrated power supply from the rack while powered on	-	-	-
Download	-	-	X
Online change	X	X	X
Clean All	-	-	X
Reset Process (IEC 60870-5-104)	-	X	X

Table 8: Variables Behavior after the Event

In lower or equal 1.5.1.0 for NX3010, NX3020 and NX3030, the retentive and persistent symbolic memories and addressable output variables memory (%Q) used to have a fixed maximum size. On table below it's possible to consult the maximum sizes allowed in these older versions.

In versions above the ones mentioned, the CPUs allow flexible retentive and persistent memory sizes. For further information, refer to the section Retain and Persistent Memory Areas.

	NX3030
Retentive addressable output variables memory (%Q)	16 Kbytes
Persistent addressable output variables memory (%Q)	48 Kbytes
Retentive symbolic variables memory	32 Kbytes
Persistent symbolic variables memory	16 Kbytes

Table 9: Retentive and Persistent memories in older versions

In the case of Clean All command, if the application has been modified so that persistent variables have been removed, inserted into the top of the list or otherwise have had its modified type, the value of these variables is lost (when prompted by the tool MasterTool to download). Thus it is recommended that changes to the persistent variables GVL only include adding new variables on the list.

Total redundant data memory: Redundant data memory is the maximum memory area that can be used as redundant memory between two redundant CPUs. This value is not a different memory, note that the sum of all redundant variables (addressable input variable, addressable output variable, addressable variable, symbolic variable, retain symbolic variable, persistent symbolic variable) must be less than or equal to the available redundant data memory.

Program memory: Program memory is the maximum size that can be used to store the user application. This area is shared with source code memory, being the total area the sum of "program memory" and "source code memory".

Source code memory (backup): This memory area is used as project backup. If the user wants to import the project, MasterTool IEC XE will get the information required in this area. Care must be taken to ensure that the project saved as a backup is up to date to avoid the loss of critical information. This area is shared with source code memory, being the total area the sum of "program memory" and "source code memory".

User files memory: This memory area offers another way for the user to store files such as doc, pdf, images, and other files. This function allows data recording as in a memory card. For further information check User Files Memory.

2.2.4. Protocols

	NX3030	Interface
Open Protocol	Yes	COM1 / COM2
MODBUS RTU Master	Yes	COM1 / COM2
MODBUS RTU Slave	Yes	COM1 / COM2
MODBUS TCP Client	Yes	NET1 / NET2
MODBUS TCP Server	Yes	NET1 / NET2
MODBUS RTU over TCP Client	Yes	NET1 / NET2
MODBUS RTU over TCP Server	Yes	NET1 / NET2
CANopen Master	No	-
CANopen Slave	No	-
CAN low level	No	-
SAE J-1939	No	-
OPC DA Server	Yes	NET1 / NET2
OPC UA Server	Yes	NET1 / NET2
EtherCAT Master	Yes	NET1 / NET2
SNMP Agent	Yes	NET1 / NET2
SOE (Event-oriented data)	Yes	NET1 / NET2
IEC 60870-5-104 Server	Yes	NET1 / NET2
EtherNet/IP Scanner	Yes	NET1 / NET2
EtherNet/IP Adapter	Yes	NET1 / NET2
MQTT Client	Yes	NET1 / NET2
SNTP Client (for clock synchronism)	Yes	NET1 / NET2
PROFINET Controller	Yes	NET1 / NET2
PROFINET Device	No	-

Table 10: Protocols

Note:

PROFINET Controller: Enabled for use without CPU redundancy and in a simple network (without a ring) with up to 8 devices. For larger applications, consult technical support.

2.2.5. Serial Interfaces

2.2.5.1. COM 1

	COM 1
Connector	Shielded female DB9
Physical interface	RS-232C
Modem signals	RTS, CTS, DCD
Baud rate	200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps
Isolation	
Logic to Serial Port	Not isolated
Serial Port to protection earth ⏚	1000 Vac / 1 minute

Table 11: COM 1 Serial Interface Features

2.2.5.2. COM 2

	COM 2
Connector	Shielded female DB9
Physical interface	RS-422 or RS-485 (depending on the selected cable)
Communication direction	RS-422: full duplex RS-485: half duplex
RS-422 max. transceivers	11 (1 transmitter and 10 receivers)
RS-485 max. transceivers	32
Termination	Yes (optional via cable selection)
Baud rate	200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps
Isolation **Logic to Serial Port** **Serial Port to protection earth** ⏚	 1000 Vac / 1 minute 1000 Vac / 1 minute

Table 12: COM 2 Serial Interface Features

Notes:

Physical interface: Depending on the configuration of the used cable, it's possible to choose the kind of physical interface: RS-422 or RS-485. The list of cables can be found at Related Products section.

RS-422 maximum transceivers: It's the maximum number of RS-422 interfaces that can be used on the same bus.

RS-485 maximum transceivers: It's the maximum number of RS-485 interfaces that can be used on the same bus.

2.2.6. Ethernet Interfaces

2.2.6.1. NET 1

	NET 1
Connector	Shielded female RJ45
Auto crossover	Yes
Maximum cable length	100 m
Cable type	UTP or ScTP, category 5
Baud rate	10/100 Mbps
Physical layer	10/100 BASE-TX (Full Duplex)
Data link layer	LLC (Logical Link Control)
Network layer	IP (Internet Protocol))
Transport layer	TCP (Transmission Control Protocol) UDP (User Datagram Protocol)
Diagnostic	LEDs - green (speed), yellow (link/activity)
Isolation **Ethernet interface to logic and earth**	 1500 Vac / 1 minute

Table 13: Ethernet NET 1 Interface Features

The NET 1 Interface is the interface used for programming using the MasterTool IEC XE tool.

2.2.6.2. NET 2

	NET 2
Connector	Shielded female RJ45
Auto crossover	Yes
Maximum cable length	100 m
Cable type	UTP or ScTP, category 5
Baud rate	10/100 Mbps
Physical layer	10/100 BASE-TX (Full Duplex)
Data link layer	LLC (Logical Link Control)
Network layer	IP (Internet Protocol)
Transport layer	TCP (Transmission Control Protocol) UDP (User Datagram Protocol)
Diagnostic	LEDs - green (speed), yellow (link/activity)
Isolation **Ethernet interface to logic and earth** **Ethernet interface to Ethernet interface**	 1500 Vac / 1 minute 1500 Vac / 1 minute

Table 14: Ethernet NET 2 Interface Features

2.2.7. Memory Card Interface

The memory card can be used for different data to be stored such as user logs, project documentation and source files.

	Memory Card
Maximum Capacity	32 Gbytes
Minimum Capacity	2 Gbytes
Type	MiniSD
File System	FAT32
Remove card safely	Yes, by pressing MS switch

Table 15: Memory Card Interface Features

Notes:

Maximum Capacity: The memory card capacity must be less than or equal to this limit for correct operation on Nexto CPU, otherwise the Nexto CPU may not detect the memory card or even present problems during data transfer.

Minimum Capacity: The memory card capacity must be greater than or equal to this limit for correct operation on Nexto CPU, otherwise the Nexto CPU may not detect the memory card or even present problems during data transfer.

File System: It is recommended to format the memory card using the Nexto CPU, otherwise it may result in performance loss in the memory card interface.

2.2.8. Environmental Characteristics

	NX3030
Current consumption on the power supply rail	1000 mA
Dissipation	5 W
Operating temperature	0 to 60 °C
Storage temperature	-25 to 75 °C
Relative humidity	5% to 96%, non-condensing
Conformal coating	Yes
IP Level	IP 20
Module dimensions (W x H x D)	36,00 x 114,63 x 115,30 mm
Package dimensions (W x H x D)	44,00 x 122,00 x 147,00 mm
Weight	350 g
Weight with package	400 g

Table 16: Environmental Characteristics

Notes:

Conformal coating of electronic circuits: The covering of electronic circuits protects internal parts of the product against moisture, dust and other harsh elements to electronic circuits.

2.3. Compatibility with Other Products

To develop an application for Nexto Series CPUs, it is necessary to check the version of MasterTool IEC XE. The following table shows the minimum version required (where the controllers were introduced) and the respective firmware version at that time:

Nexto Series CPUs	MasterTool IEC XE	Firmware version
NX3010, NX3020, NX3030	1.00 to 2.09	1.2.0.9 to 1.7.0.14
NX3010, NX3020, NX3030	3.00 or above	1.8.3.0 or above

Table 17: Compatibility with other products

Additionally, along the development roadmap of MasterTool IEC XE some features may be included (like special Function Blocks, etc...), which can introduce a requirement of minimum firmware version. During the download of the application, MasterTool IEC XE checks the firmware version installed on the controller and, if it does not meets the minimum requirement, will show a message requesting to update. The latest firmware version can be downloaded from Altus website, and it is fully compatible with previous applications.

2.4. Performance

The Nexto Series CPUs performance relies on:

- User Application Time
- Application Interval
- Operational System Time
- Module quantity (process data, input/output, among others)

2.4.1. MainTask Interval Time

The MainTask interval time setting depends on the selected project profile. For the profiles Simple, Normal, Experienced, and Custom profiles, the interval can be set with values from 1 ms to 750 ms. For the Machine Machine Profile, the interval can be configured with values from 1 ms to 100 ms.

2.4.2. Application Times

The execution time of Nexto CPUs application depends on the following variables:

- Input read time (local and remote)
- Tasks execution time
- Output write time (local and remote)

It is important to stress that the execution time of the "MainTask" will be directly influenced by the "Configuration" system task, a task of high priority, executed periodically by the system. The "Configuration" task may interrupt the "MainTask" and, when using the communication modules, as the Ethernet NX5000 module, for instance, the time addition to the "MainTask" may be up to 25% of the execution average time.

2.4.3. Time for Instructions Execution

The table below presents the necessary execution time for different instructions.

Instruction	Language	Variables	Instruction Times (μs)
1000 Contacts	LD	BOOL	6
1000 Divisions	ST	INT	43
		REAL	81
	LD	INT	43
		REAL	81
1000 Multiplications	ST	INT	15
		REAL	23
	LD	INT	15
		REAL	23
1000 Sums	ST	INT	15
		REAL	23
	LD	INT	15
		REAL	23
1000 PID	ST	REAL	< 5000

Table 18: Instruction Times

2.4.4. Initialization Times

Nexto Series CPUs have initialization times of 50 s, and the initial screen with the NEXTO logo (Splash) is presented after 20 s from the power switched on.

2.5. Physical Dimensions

Dimensions in mm.

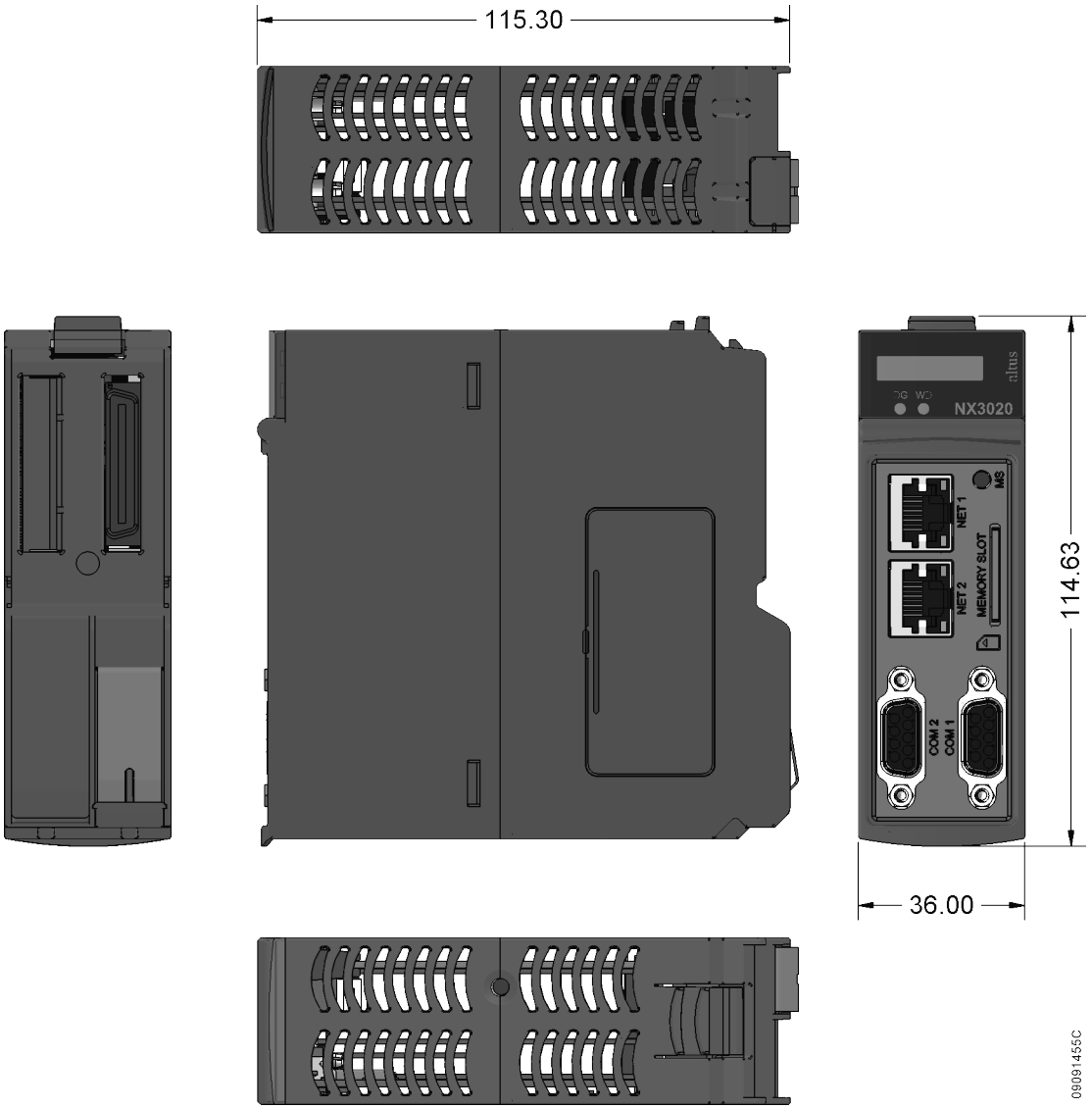


Figure 4: CPU Physical Dimensions

2.6. Purchase Data

2.6.1. Included Itens

The product package contains the following items:

- NX3030 module

2.6.2. Product code

The following code should be used to purchase the product:

Code	Description
NX3030	High-speed CPU, 2 Ethernet ports, 2 serial channels, memory card interface, remote rack expansion and redundancy support

Table 19: Product Code

2.7. Related Products

The following products must be purchased separately when necessary:

Code	Description
MT8500	MasterTool IEC XE
AL-2600	RS-485 network branch and terminator
AL-2306	RS-485 cable for MODBUS or CAN network
AL-2319	RJ45-RJ45 Cable
AL-1729	RJ45-CMDB9 Cable
AL-1748	CMDB9-CFDB9 Cable
AL-1752	CMDB9-CMDB9 Cable
AL-1753	CMDB9-CMDB25 Cable
AL-1754	CMDB9-CFDB9 Cable
AL-1761	CMDB9-CMDB9 Cable
AL-1762	CMDB9-CMDB9 Cable
AL-1763	CMDB9-Terminal Block Cable
AL-1766	CFDB9-Terminal Block Cable
NX9101	32 GB microSD memory card with miniSD and SD adapters
NX9202	RJ45-RJ45 2 m Cable
NX9205	RJ45-RJ45 5 m Cable
NX9210	RJ45-RJ45 10 m Cable
NX9000	8-Slot Backplane Rack
NX9001	12-Slot Backplane Rack
NX9002	16-Slot Backplane Rack
NX9003	24-Slot Backplane Rack
NX8000	30 W 24 Vdc Power Supply Module

Table 20: Related Products

Notes:

MT8500: MasterTool IEC XE is available in four different versions: LITE, BASIC, PROFESSIONAL and ADVANCED. For more details, please check MasterTool IEC XE User Manual - MU299609.

AL-2600: This module is used for branch and termination of RS-422/485 networks. For each network node, an AL-2600 is required. The AL-2600 that is at the ends of network must be configured with termination, except when there is a device with active internal termination, the rest must be configured without termination.

AL-2306: Two shielded twisted pairs cable without connectors, used for networks based on RS-485 or CAN.

AL-2319: Two RJ45 connectors for programming the CPUs of the Nexto Series and Ethernet point-to-point with another device with Ethernet interface communication.

AL-1729: RS-232C standard cable with one RJ45 connector and one DB9 male connector for communication between CPUs of the Nexto Series and other Altus products of the DUO Series, Piccolo Series and Ponto Series.

AL-1748: RS-232C standard cable with one DB9 male connector and one DB9 female connector for communication between CPUs of the Nexto Series and Altus products of the Cimrex Series.

AL-1752: RS-232C standard cable with two DB9 male connectors for communication between CPUs of the Nexto Series and Altus products of the H Series and iX series.

AL-1753: RS-232C standard cable with one DB9 male connector and one DB25 male connector for communication between CPUs of the Nexto Series and Altus products of the H Series.

AL-1754: RS-232C standard cable with one DB9 male connector and one DB9 female connector for communication between CPUs of the Nexto Series and Altus products of the Exter Series or Serial port, RS-232C standard, of a microcomputer.

AL-1761: RS-232C standard cable with two DB9 male connectors for communication between Nexto Series CPUs and Altus products of the AL Series.

AL-1762: RS-232C standard cable with two DB9 male connectors for communication between Nexto Series CPUs.

AL-1763: Cable with one DB9 male connector and terminal block for communication between CPUs of the Nexto Series and products with RS-485/RS-422 standard terminal block.

AL-1766: Cable with a female DB9 connector and terminals for communication between HMI P2 and Nexto Xpress/NX3003 controllers.

NX9202/NX9205/NX9210: Cables used for Ethernet communication and to interconnect the bus expansion modules.

3. Installation

This chapter presents the necessary proceedings for the Nexto Series CPUs physical installation, as well as the care that should be taken with other installation within the panel where the CPU is been installed.

CAUTION

If the equipment is used in a manner not specified by in this manual, the protection provided by the equipment may be impaired.

3.1. Mechanical Installation

Nexto Series CPUs must be inserted in the backplane rack position 2, just beside the Power Supply Module. All information regarding mechanical installation and module insertion can be found at MU214600 - Nexto Series User Manual .

3.2. Electrical Installation

DANGER

When executing any installation in an electric panel, certify that the main energy supply is OFF.

The CPUs energy supply come from the Power Supply Module which supplies the CPUs power through the backplane rack connection. It does not need any external connection. The module grounding is given through the contact between the module grounding spring and the backplane rack.

The figure below shows the Nexto Series CPUs electric diagram installed in a Nexto Series backplane rack.

The connectors placement depicted are merely illustrative.

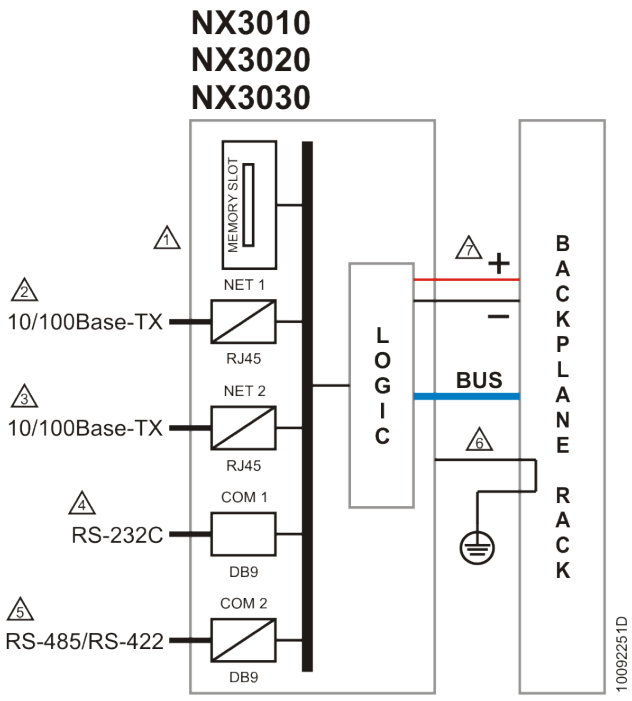


Figure 5: NX3010, NX3020 and NX3030 CPUs Electric Diagram

Diagram Notes:

1 Memory card interface.

2 Ethernet interface 10/100Base-TX standard for programming, debugging and MODBUS TCP network connection or other protocols.

3 Ethernet interface 10/100Base-TX standard for MODBUS TCP network connection or other protocols (only for NX3020 and NX3030).

4 Serial interface RS-232C standard for MODBUS RTU network connection or other protocols.

5 Serial interface RS-485/RS-422 standard for MODBUS RTU network connection or other protocols. The physical interface choice depends on the cable used.

6 The module is grounded through Nexto Series backplane rack.

7 The power supply comes from the backplane rack connection. There is no need for external connections.

Protection earth terminal.

3.3. Ethernet Network Connection

The NET 1 and NET 2 isolated communication interface allows the connection with an Ethernet network, however, the NET 1 interface is the most suitable to be used for communication with MasterTool IEC XE.

The Ethernet network connection uses twisted pair cables (10/100Base-TX) and the speed detection is automatically made by the Nexto CPU. This cable must have one of its endings connected to the interface that is likely to be used and another one to the HUB, switch, microcomputer or other Ethernet network point.

3.3.1. IP Address

The NET 1 Ethernet interface is used for Ethernet communication and for CPU configuration which comes with the following default parameters configuration:

	NET 1
IP Address	192.168.15.1
Subnetwork Mask	255.255.255.0
Gateway Address	192.168.15.253

Table 21: Default Parameters Configuration for Ethernet NET 1 Interface

The IP Address and Subnet Mask parameters can be seen on the CPU graphic display via parameters menu, as described in CPU's Informative and Configuration Menu section.

Initially, the NET 1 interface must be connected to a PC network with the same subnet mask to communicate with MasterTool IEC XE, where the network parameters can be modified. For further information regarding configuration and parameters modifications, see Ethernet Interfaces Configuration section.

The NET 2 Ethernet interface is used only for Ethernet communication and comes with the following default parameters configuration:

	NET 2
IP Address	192.168.16.1
Subnetwork Mask	255.255.255.0
Gateway Address	192.168.16.253

Table 22: Default Parameters Configuration for Ethernet NET 2 Interface

The IP Address and Subnet Mask parameters can be seen on the CPU graphic display via parameters menu, as described in CPU's Informative and Configuration Menu section.

The NET 2 interface network parameters can be changed through MasterTool IEC XE. For further information regarding configuration and parameters modifications, see Ethernet Interfaces Configuration section.

3.3.2. Gratuitous ARP

The NETx Ethernet interface promptly sends ARP packets type in broadcast informing its IP and MAC address for all devices connected to the network. These packets are sent during a new application download by the MasterTool IEC XE software and in the CPU startup when the application goes into Run mode.

Five ARP commands are triggered within a 200 ms initial interval, doubling the interval every new triggered command, totalizing 3 s. Example: first trigger occurs at time 0, the second one at 200 ms and the third one at 600 ms and so on until the fifth trigger at time 3 s.

3.3.3. Network Cable Installation

Nexto Series CPUs Ethernet ports, identified on the panel by NET, have standard pinout which are the same used in PCs. The connector type, cable type, physical level, among other details regarding the CPU and the Ethernet network device are defined in the Ethernet Interfaces.

The table below present the RJ-45 Nexto CPU female connector, with the identification and description of the valid pinout for 10BASE-TE and 100BASE-TX physical levels.

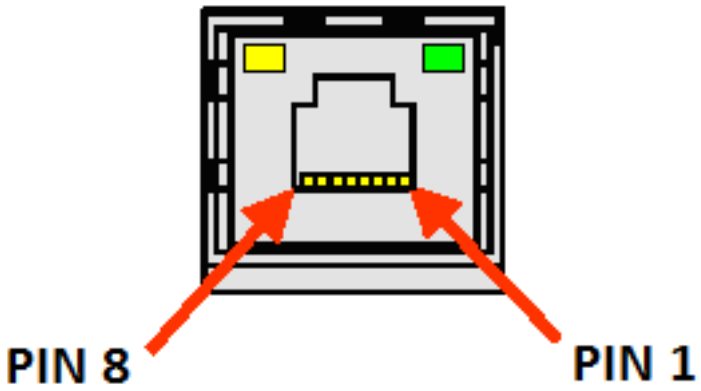


Figure 6: RJ45 Female Connector

Pin	Signal	Description
1	TXD +	Data transmission, positive
2	TXD -	Data transmission, negative
3	RXD +	Data reception, positive
4	NU	Not used
5	NU	Not used
6	RXD -	Data reception, negative
7	NU	Not used
8	NU	Not used

Table 23: RJ45 Female Connector Pinout - 10BASE-TE and 100BASE-TX

The interface can be connected in a communication network through a hub or switch, or straight from the communication equipment. In this last case, due to Nexto CPUs Auto Crossover feature, there is no need for a cross-over network cable, the one used to connect two PCs point to point via Ethernet port.

It is important to stress that it is understood by network cable a pair of RJ45 male connectors connected by a UTP or ScTP cable, category 5 whether straight connecting or cross-over. It is used to communicate two devices through the Ethernet port.

These cables normally have a connection lock which guarantees a perfect connection between the interface female connector and the cable male connector. At the installation moment, the male connector must be inserted in the module female connector until a click is heard, assuring the lock action. To disconnect the cable from the module, the lock lever must be used to unlock one from the other.

3.4. Serial Network Connection RS-232

The COM 1 non isolated communication interface allows the connection to a RS-232C network. As follows it's presented the DB9 female connector to Nexto CPU, with identification and sign description.

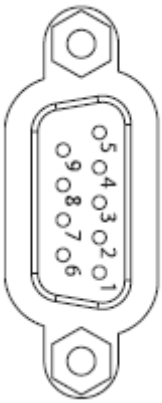


Figure 7: DB9 Female Connector

Pin	Sign	Description
1	DCD	Data Carrier Detect
2	TXD	Data Transmission
3	RXD	Data Reception
4	-	Not used
5	GND	Ground
6	-	Not used
7	CTS	Clear to Send
8	RTS	Request to Send
9	-	Not used

Table 24: COM 1 DB9 Female Connector Pin Layout

3.4.1. RS-232C Communication

For connection to a RS-232C device, use the appropriate cable as the section Related Products.

3.5. Serial Network Connection RS-485/422

The COM 2 isolated communication interface allow the connection to a RS-485/422 network. As follows it's presented the DB9 female connector to Nexto CPU, with identification and sign description.

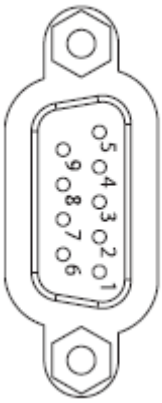


Figure 8: DB9 Female Connector

Pin	Sign	Description
1	-	Not used
2	Term+	Internal Termination, positive
3	TXD+	Data Transmission, positive
4	RXD+	Data Reception, positive
5	GND	Negative Reference for External Termination
6	+5V	Positive Reference for External Termination
7	Term-	Internal Termination, negative
8	TXD-	Data Transmission, negative
9	RXD-	Data Reception, negative

Table 25: COM 1 and COM 2 DB9 Female Connector Pin Layout

3.5.1. RS-485 Communication without termination

In order to connect in a RS-485 network with no termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 terminals	Device terminal signals
0	Shield
1	Not connected
2	D+
3	D+
4	Not connected
5	Not connected
6	Not connected
7	D-
8	D-

Table 26: RS-485 Connections without Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.

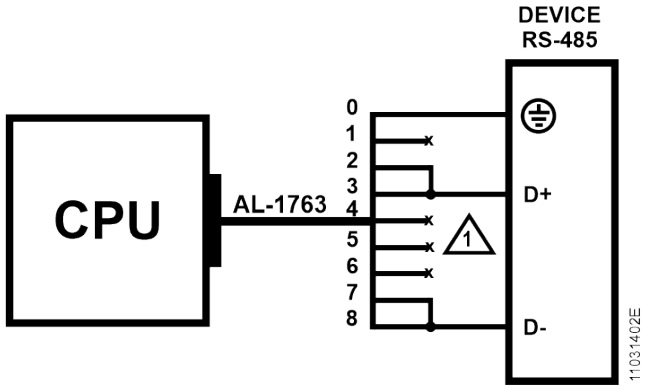


Figure 9: RS-485 Connections without Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.2. RS-485 Communication with Internal Termination

In order to connect in a RS-485 network using the internal termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 terminals	CPU terminal signals
0	Shield
1	D+
2	D+
3	D+
4	Not connected
5	Not connected
6	D-
7	D-
8	D-

Table 27: RS-485 Connections with Internal Termination

PS.: The internal termination available is a safe state type in open mode.

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.

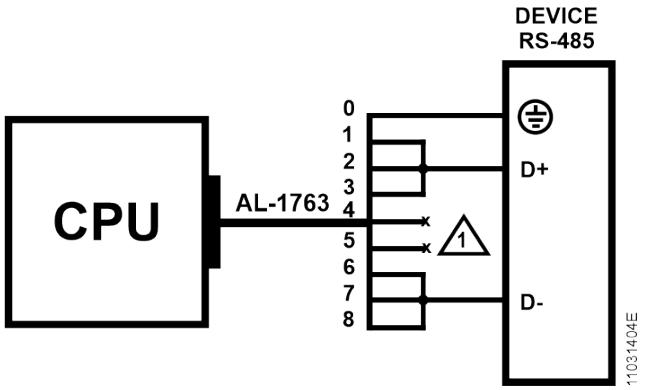


Figure 10: RS-485 Connections with Internal Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.3. RS-485 Communication with External Termination

In order to connect to a RS-485 network wih external termination, the AL-1763 cable identified terminals must be connected in the respective device terminals according to the table below.

AL-1763 terminals	CPU terminal signals
0	Shield
1	Not connected
2	D+
3	D+
4	0 V
5	+5 V
6	Not connected
7	D-
8	D-

Table 28: RS-485 Connections with External Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.

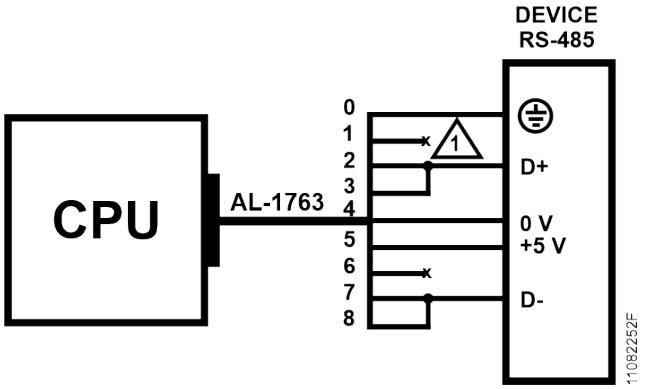


Figure 11: RS-485 Connections with External Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.4. Example of Connection of a RS-485 Network with External Termination and Master Redundancy

The figure below shows an example of RS-485 network connection with external termination, using two Nexto NX3030 CPUs with half-cluster redundancy as master.

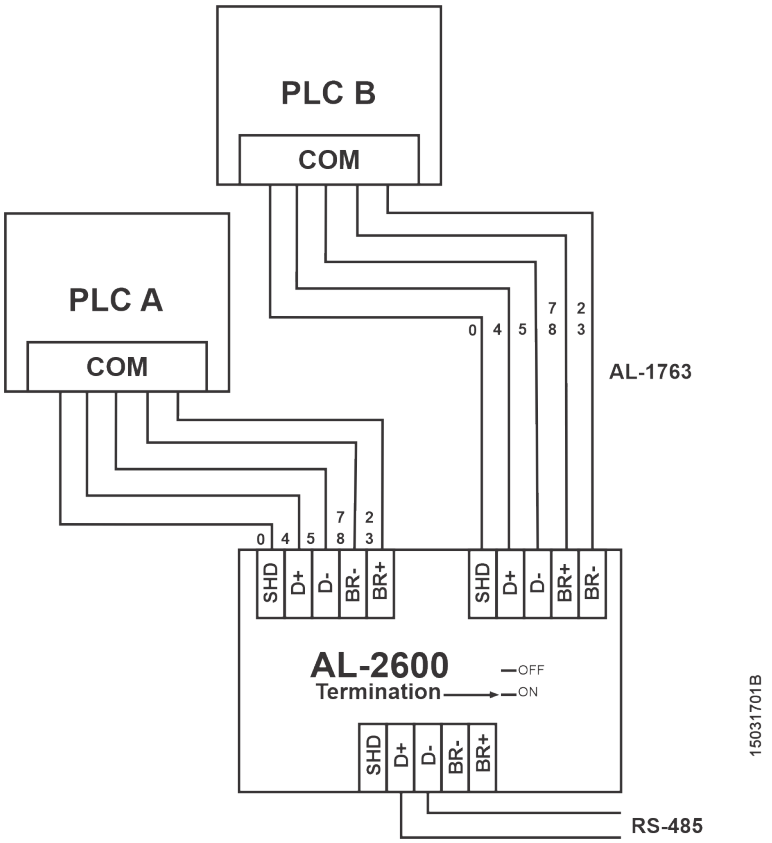


Figure 12: Connection Diagram of a RS-485 Network with External Termination and Master Redundancy

3.5.5. RS-422 Communication without Termination

In order to connect in a RS-422 network with no termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 terminals	CPU terminal signals
0	Shield
1	Not connected
2	TX+
3	RX+
4	Not connected
5	Not connected
6	Not connected
7	TX-
8	RX-

Table 29: RS-422 Connections without Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.

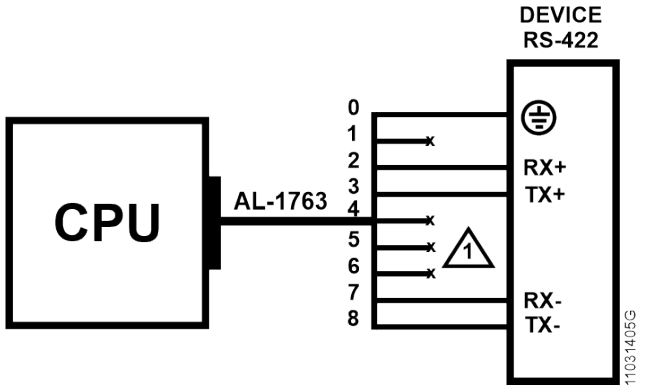


Figure 13: Connections without Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.6. RS-422 Communication with Internal Termination

In order to connect in a RS-422 network using the internal termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 terminals	CPU terminal signals
0	Shield
1	TERM+
2	TX+
3	RX+
4	Not connected
5	Not connected
6	TERM-
7	TX-
8	RX-

Table 30: RS-422 Connections with Internal Termination

PS.: The internal terminations available are secure state in open mode.

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.

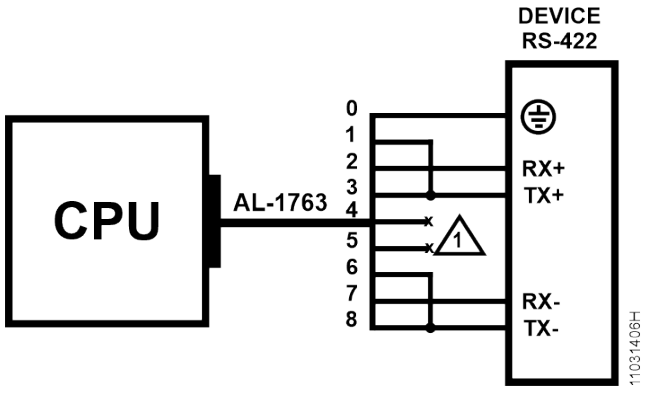


Figure 14: RS-422 Connections with Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.7. RS-422 Communication with External Termination

In order to connect in a RS-422 network using interface external termination, the cable AL-1763 identified terminals must be connected in the respective device terminals, as shown on table below.

AL-1763 Terminals	CPU terminal signals
0	Shield
1	Not connected
2	TX+
3	RX+
4	0 V
5	+5 V
6	Not connected
7	TX-
8	RX-

Table 31: RS-422 Connections with External Termination

The figure diagram below indicates how the AL-1763 connection terminals should be connected in the device terminals.

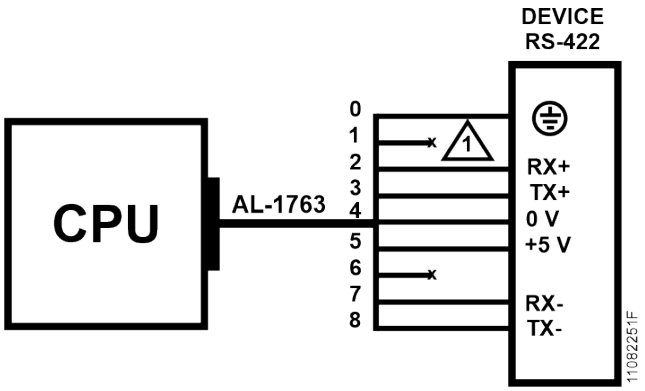


Figure 15: RS-422 Connections with External Termination Diagram

Diagram Note:

1. The not connected terminals must be insulated so they do not make contact with each other.

3.5.8. RS-422 Network Example

The figure below shows an example of RS-422 network utilization, using the Nexto CPU as master, slave devices with RS-422 Interface, and Altus solutions for terminators and connections.

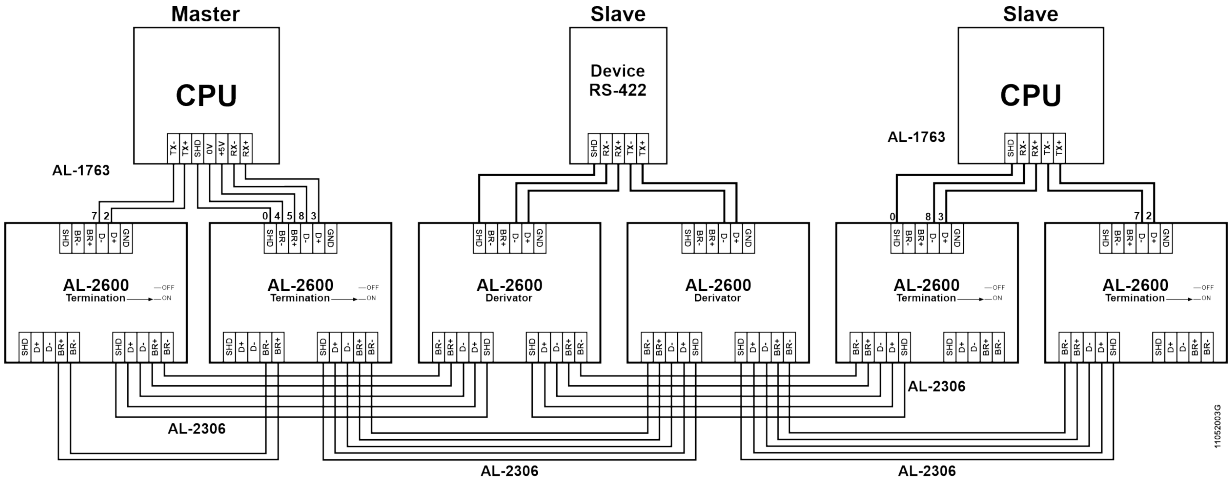


Figure 16: RS-422 Network Example

Diagram Note:

The AL-2600 modules which are in the network endings perform the terminators function. In this case the AL-2600 keys must be configured in PROFIBUS Termination.

3.6. Memory Card Installation

This section presents how to insert the memory card into the models Nexto Series CPUs. For further information see Memory Card section.

Initially, care must be taken with the correct position the memory card must be inserted. One corner of it is different from the other three and this one must be used as reference for the card correct insertion. Therefore, the memory card must be inserted following the depiction on the CPU frontal part or the way showed on figure below.

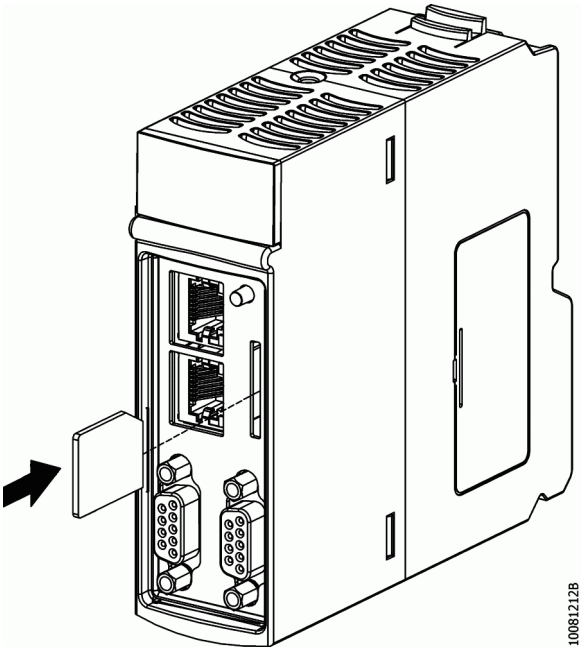


Figure 17: Memory Card Insertion in the CPU

When the card is correctly installed, a symbol will appear on the CPU graphic display. For card secure removing the MS key must be pressed then there is a little delay and the card symbol will disappear from the graphic display. The card is now

ready to be taken off. For that, the card must be pressed against the CPU until a click is heard, then release it and withdraw it from the compartment as showed on figure below. At this moment the card will be loose.

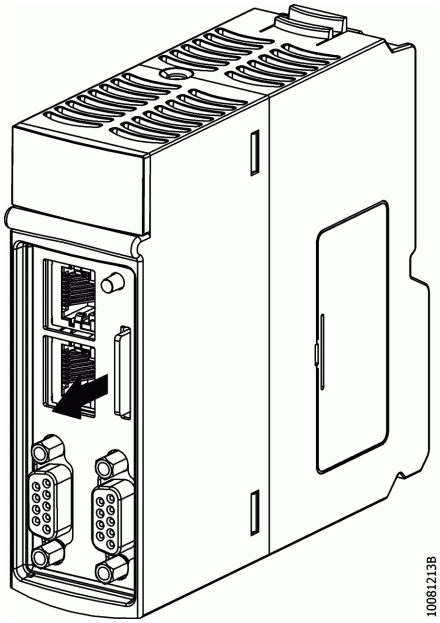


Figure 18: Memory Card Withdrawal

3.7. Architecture Installation

3.7.1. Module Installation on the Main Backplane Rack

Nexto Series has an exclusive method for connecting and disconnecting modules on the bus which does not require much effort from the operator and guarantee the connection integrity. For further information regarding Nexto Series products fixation, please see Nexto Series User Manual – MU214600.

3.8. Programmer Installation

To execute the MasterTool IEC XE development software installation, it is necessary to have the distribution CD-ROM or download the installation file from the site https://www.altus.com.br/en/. For further information about the step by step to installation, consult MasterTool IEC XE User Manual MT8500 – MU299609.

4. Initial Programming

The main goal of this chapter is to help the programming and configuration of Nexto Series CPUs, allowing the user to take the first steps before starting to program the device.

Nexto Series CPU uses the standard IEC 61131-3 for language programming, which are: IL, ST, LD, SFC and FBD, and besides these, an extra language, CFC. These languages can be separated in text and graphic. IL and ST are text languages and are similar to Assembly and C, respectively. LD, SFC, FBD and CFC are graphic languages. LD uses the relay block representation and it is similar to relay diagrams. SFC uses the sequence diagram representation, allowing an easy way to see the event sequence. FBD and CFC use a group of function blocks, allowing a clear vision of the functions executed by each action.

The programming is made through the MasterTool IEC XE (IDE) development interface. The MasterTool IEC XE allows the use of the six languages in the same project, so the user can apply the best features offered by each language, resulting in more efficient applications development, for easy documentation and future maintenance.

For further information regarding programming, see MasterTool IEC XE User Manual - MU299609, MasterTool IEC XE Programming Manual - MP399609 or IEC 61131-3 standard.

4.1. Memory Organization and Access

Nexto Series uses an innovative memory organization and access feature called big-endian, where the most significant byte is stored first and will always be the smallest address (e.g. %QB0 will always be more significant than %QB1, as in table below, where, for CPUNEXTO string, the letter C is byte 0 and the letter O is the byte 7).

Besides this, the memory access must be done carefully as the variables with higher number of bits (WORD, DWORD, LONG), use as index the most significant byte, in other words, the %QD4 will always have as most significant byte the %QB4. Therefore it will not be necessary to make calculus to discover which DWORD correspond to defined bytes. The table below, shows little and big endian organization.

MSB ← Little-endian → LSB								
BYTE	%QB7	%QB6	%QB5	%QB4	%QB3	%QB2	%QB1	%QB0
	C	P	U	N	E	X	T	O
WORD	%QW6		%QW4		%QW2		%QW0	
	CP		UN		EX		TO	
DWORD	%QD4				%QD0			
	CPUN				EXTO			
LWORD	%QL0							
	CPUNEXTO							
MSB ← Big-endian → LSB								
BYTE	%QB0	%QB1	%QB2	%QB3	%QB4	%QB5	%QB6	%QB7
	C	P	U	N	E	X	T	O
WORD	%QW0		%QW2		%QW4		%QW6	
	CP		UN		EX		TO	
DWORD	%QD0				%QD4			
	CPUN				EXTO			
LWORD	%QL0							
	CPUNEXTO							

Table 32: Memory Organization and Access Example

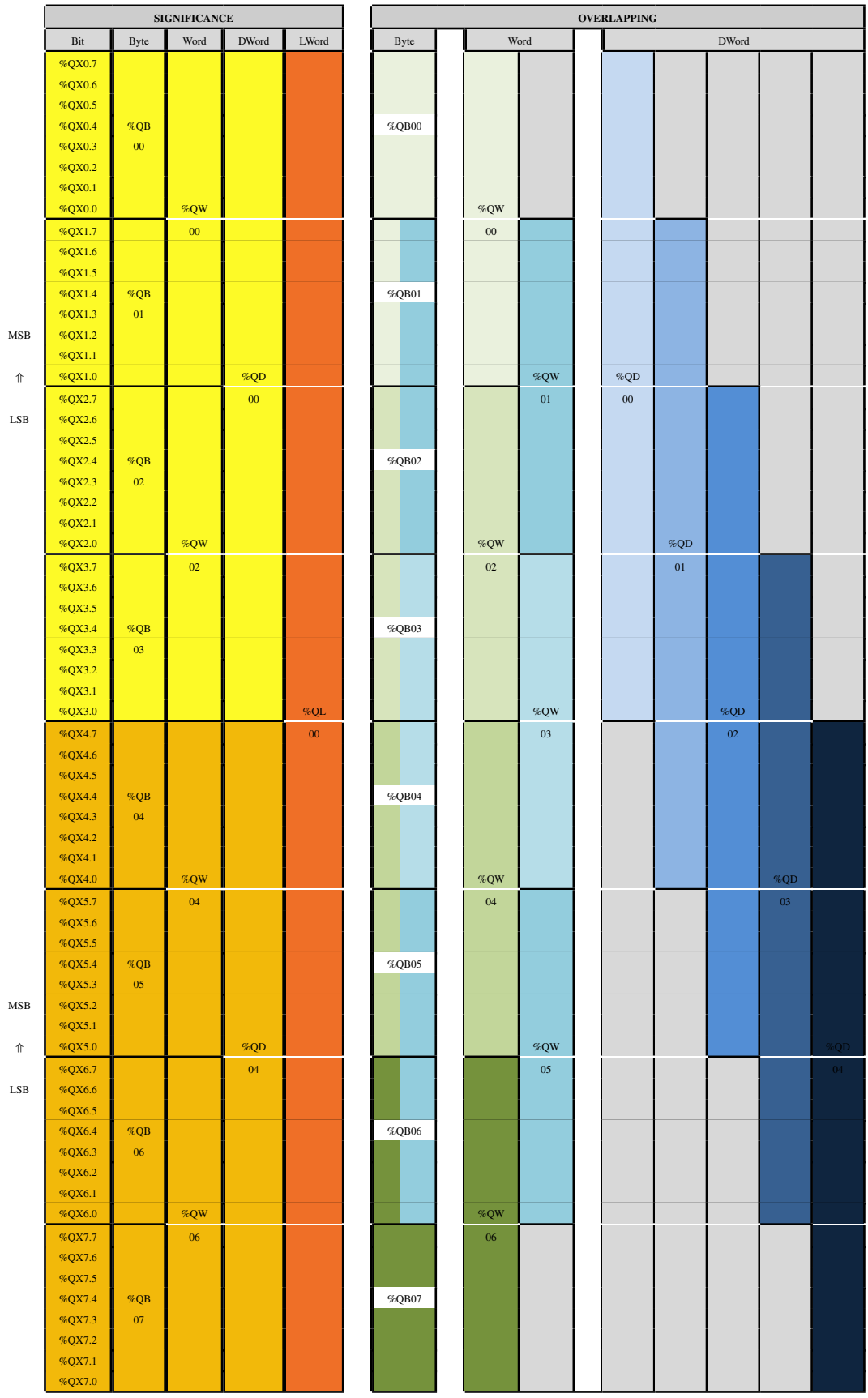


Table 33: Memory Organization and Access

The table above shows the organization and memory access, illustrating the significance of bytes and the disposition of other variable types, including overlapping.

4.2. Project Profiles

A project profile in the MasterTool IEC XE consists in an application template together with a group of verification rules which guides the development of the application, reducing the programming complexity. The applications can be created according the following profiles:

- Single
- Basic
- Normal
- Expert
- Custom
- Machine Profile

The Project Profile is selected on the project creation wizard. Each project profile defines a template of standard names for the tasks and programs, which are pre-created according to the selected Project Profile. Also, during the project compilation (generate code), MasterTool IEC XE verify all the rules defined by the selected profile.

The following sections details the characteristics of each profile, which follow a gradual complexity slope. Based in these definitions, it's recommended that the user always use the simplest profile that meets his application needs, migrating to a more sophisticated profile only when the corresponding rules are being more barriers to development than didactic simplifications. It is important to note that the programming tool allows the profile change from an existent project (see project update section in the MasterTool IEC XE User Manual – MU299609), but it's up to the developer to make any necessary adjustments so that the project becomes compatible with the rules of the new selected profile.

ATTENTION

Through the description of the Project profiles some tasks types are mentioned, which are described in the section 'Task Configuration', of the MasterTool IEC XE User Manual – MU299609.

ATTENTION

When more than one task is used, the I/O access can only be done in the context of the MainTask. In case that the option Enable I/O Update per Task can't be used, present as of MasterTool IEC XE version 2.01.

4.2.1. Single

In the Single Project Profile, the application has only one user task, MainTask. This task is responsible for the execution of a single Program type programming unit called MainPrg. This single program can call other programming unit, of the Program, Function or Function Block types, but the whole code will be executed exclusively by the MainTask.

In this profile, the MainTask will be of the cyclical type (Cyclic) with priority fixed as 13 (thirteen) and runs exclusively the MainPrg program in a continuous loop. The MainTask is already fully defined and the developer needs to create the MainPrg program, using any of the languages of the IEC 61131-3 standard. It is not always possible to convert a program to another language, but it's always possible to create a new program, built in a different language, with the same name and replace it. The MasterTool IEC XE standard option is to use the MasterTool Standard Project associated with the Single profile, which also include the MainPrg created in the language selected during the project creation.

This type of application never needs to consider issues as data consistence, resource sharing or mutual exclusion mechanisms.

Task	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	20 ms	-

Table 34: Single Profile Task

4.2.2. Basic

In the Basic Project Profile, the application has one user task of the Continuous type called MainTask, which executes the program in a continuous loop (with no definition of cycle time) with priority fixed in 13 (thirteen). This task is responsible for the execution of a single programming unit POU called MainPrg. It's important to notice that the cycle time may vary according to the quantity of communication tasks used, as in this mode, the main task is interrupted by communication tasks.

This profile also allows the inclusion of two event tasks with higher priority, that can interrupt (preempt) the MainTask at any given moment: the task named ExternInterruptTask00 is an event task of the External type with priority fixed in 02 (two); the task named TimeInterruptTask00 is an event task of the Cyclic type with priority fixed as 01 (one).

The Basic project template model includes three tasks already completely defined as presented in table below. The developer need only to create the associated programs.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Continuous	-	-
ExternInterruptTask00	ExternInterruptPrg00	02	External	-	IO_EVT_0
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	20 ms	-

Table 35: Basic Profile Tasks

4.2.3. Normal

In the Normal Project Profile, the application has one user task of the Cyclic type, called MainTask. This task is responsible for the execution of a single programming unit POU called MainPrg. This program and this task are similar to the only task and only program of the Single profile, but here the application can integrate additional user tasks. These other tasks, named CyclicTask00 and CyclicTask01, each one responsible for the exclusive execution of its respective CyclicPrg<nn> program. The CyclicTask<nn> tasks are always of the cyclic type and with priority fixed in 13 (thirteen), same priority as MainTask. These two types form a group called basic tasks, which associated programs can call other POUs of the Program, Function and Function Block types.

Furthermore, this profile can include event tasks with higher priority than the basic tasks, which can interrupt (preempt) these tasks execution at any time.

The task called ExternInterruptTask00 is an event task of the External type which execution is triggered by some external event, such as the variation of a control signal on a serial port or the variation of a digital input on the NEXTO bus. This task priority is fixed in 02 (two), being responsible exclusively for the execution of the ExternInterruptPrg00 program. The task called TimeInterruptTask00 is an event task of the Cyclic type with a priority fixed as 01 (one), being responsible for the execution exclusively of TimeInterruptPrg00 program.

In the Normal project model, there are five tasks, and its POUs, already fully defines as shown in table below. The developer needs only to implement the programs content, opting, on the wizard, for any of the languages in IEC 61131-3 standard. The tasks interval and trigger events can be configured by the developer and the unnecessary tasks can be eliminated.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	20 ms	-
CyclicTask00	CyclicPrg00	13	Cyclic	200 ms	-
CyclicTask01	CyclicPrg01	13	Cyclic	500 ms	-
ExternInterruptTask00	ExternInterruptPrg00	02	External	-	IO_EVT_0
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	20 ms	-

Table 36: Normal Profile Tasks

4.2.4. Expert

The Expert Project Profile includes the same basic tasks, CyclicTask<nn>, ExternInterruptTask00 and TimeInterruptTask00 with the same priorities (13, 02 and 01 respectively), but it's an expansion from the previous ones, due to accept multiple events tasks. That is, the application can include various ExternInterruptTask<nn> or TimeInterruptTask<nn> tasks that execute the ExternInterruptPrg<nn> and TimeInterruptPrg<nn> programs. The additional event tasks priorities can be freely selected from 08 to 12. In this profile, besides the standard programs, each task can execute additional programs.

In this project profile, the application may also include the user task FreeTask of the Freewheeling type with priority 31, responsible for the FreePrg program execution. As this task is low priority it can be interrupted by all others so it can execute codes that might be blocked.

There are eight tasks already fully defined, as shown in table below, as well as their associated programs in the chosen language. Intervals and trigger events of any task, as well as the priorities of the event tasks can be configured by the user.

When developing the application using Expert project's profile, a special care is needed with the event tasks scaling. If there is information and resource sharing between these tasks or between them and the basic tasks, it is strongly recommended to adopt strategies to ensure data consistency.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	20 ms	-
CyclicTask00	CyclicPrg00	13	Cyclic	200 ms	-
CyclicTask01	CyclicPrg01	13	Cyclic	500 ms	-
ExternInterruptTask00	ExternInterruptPrg00	02	External	-	IO_EVT_0
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	20 ms	-
ExternInterruptTask01	ExternInterruptPrg01	11	External	-	IO_EVT_1
TimeInterruptTask01	TimeInterruptPrg01	09	Cyclic	30 ms	-
FreeTask	FreePrg	31	Continuous	-	-

Table 37: Expert Profile Tasks

4.2.5. Custom

The Custom project profile allows the developer to explore all the potential of the Runtime System implemented in the CPUs. No functionality is disabled; no priority, task and programs association or nomenclatures are imposed. The only exception is for MainTask, which must always exist with this name in this Profile.

Beyond the real time tasks, with priority between 00 and 15, which are scheduled by priority, in this profile it is also possible to define tasks with lower priorities in the range 16 to 31. In this range, it's used the Completely Fair Scheduler (time sharing), which is necessary to run codes that can be locked (for example, use of sockets).

The developer is free to partially follow or not the organization defined in other project profiles, according to the characteristics of the application. On the other hand, the Custom model associated with this profile needs no pre-defining elements such as task, program or parameter, leaving the developer to create all the elements that make up the application.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	20 ms	-
CyclicTask00	CyclicPrg00	13	Cyclic	200 ms	-
CyclicTask01	CyclicPrg01	13	Cyclic	500 ms	-
ExternInterruptTask00	ExternInterruptPrg00	02	External	-	IO_EVT_0
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	20 ms	-
ExternInterruptTask01	ExternInterruptPrg01	11	External	-	IO_EVT_1
TimeInterruptTask01	TimeInterruptPrg01	09	Cyclic	30 ms	-
FreeTask	FreePrg	31	Continuous	-	-

Table 38: Custom Profile Tasks

4.2.6. Machine Profile

In the Machine Profile, by default, the application has a user task of the Cyclic type called MainTask. This task is responsible for implementing a single Program type POU called MainPrg. This program can call other programming units of the Program, Function or Function Block types, but any user code will run exclusively by MainTask.

This profile is characterized by allowing shorter intervals in the MainTask, allowing faster execution of user code. This optimization is possible because MainTask also performs the processing of the bus. This way, different from other profiles, the machine profile requires no context switch for the bus treatment, which reduces the overall processing time.

This profile may further include an interruption task, called TimeInterruptTask00, with a higher priority than the MainTask, and hence, can interrupt its execution at any time.

Tasks	POU	Priority	Type	Interval	Event
MainTask	MainPrg	13	Cyclic	20 ms	-
TimeInterruptTask00	TimeInterruptPrg00	01	Cyclic	4 ms	-

Table 39: Machine Profile Tasks

Also, this profile supports the inclusion of additional tasks associated to the external interruptions.

4.2.7. General Table

		Project Profiles					
		Single	**Machine**	**Basic**	**Normal**	**Expert**	**Custom**
Total tasks		01	04	[01..03]	[01..32]	[01..32]	[01..32]
Tasks per program		01		01	01	<n>	<n>
Main Task	**Type**	Cyclic	Cyclic	Continuous	Cyclic	Cyclic	Cyclic
	Priority	13	13	13	13	13	13
	Quantity	01	01	01	01	01	01
Time Interrupt Task	**Type**		Cyclic	Cyclic	Cyclic	Cyclic	Cyclic
	Priority		01	01	01	01 or [08..12]	01 or [08..12]
	Quantity		[00..01]	[00..01]	[00..01]	[00..31]	[00..31]
Extern Interrupt Task	**Type**		External	External	External	External	External
	Priority		02	02	02	02 or [08..12]	02 or [08..12]
	Quantity		[00..01]	[00..01]	[00..01]	[00..31]	[00..31]
Ciclic Task	**Type**				Cyclic	Cyclic	Cyclic
	Priority				13	13	13
	Quantity				[00..31]	[00..31]	[00..31]
Free Task	**Type**					Continuous	Continuous
	Priority					31	31
	Quantity					[00..01]	[00..01]
Event Task	**Type**						Event
	Priority						<n>
	Quantity						[00..31]

Table 40: General Profile x Tasks Table

ATTENTION

The suggested POU names associated with the tasks are not consisted. They can be changed, as long as they are also changed in the tasks configurations.

4.2.8. Maximum Number of Tasks

The maximum number of tasks that the user can create is only defined for the Custom profile, the only one which has this permission. The others already have their tasks created and configured. However, the tasks that will be created must use the following prefixes, according to the type of each of the tasks: CyclicTaskxx, TimeInterruptTaskxx, ExternInterruptTaskxx, where xx represents the number of the task that being created.

The table below describes the maximum IEC task quantity per CPU and project profile, where the protocol instances are also considered communication tasks by the CPU.

	Task Type	NX3030					
		S	B	N	E	P	M
Configuration Task (WHSB Task)	Cyclic	1	1	1	1	1	0
User Tasks	Cyclic	1	1	31	31	31	2
	Triggered by Event	0	0	0	0	31	0
	Disp. External Event	0	1	0	30	31	0
	Freewheeling	0	1	0	1	31	0
	State-triggered	0	0	0	0	31	0
NETs - Client or Server Instances	Cyclic	16					
COM (n) - Master or Slave Instances	Cyclic	1					
TOTAL		32					

Table 41: NX3030 IEC Tasks Maximum Number

Notes:

Profile Legend: The S, B, N, E, C and M letters correspond to the Single, Basic, Normal, Expert, Custom and Machine profiles respectively.

Values: The number defined for each task type represents the maximum values allowed.

Task WHSB: The WHSB is a system task that must be considered so the total value is not surpassed.

NETs - Client or Server Instances: The maximum value defined considers all system Ethernet interfaces, including the expansion modules when these are applied. E.g. MODBUS protocol instances.

COM (n) - Master or Slave Instances: The "n" represents the number of the serial interface. Even with expansion modules, the table value will be the maximum per interface. E.g. MODBUS protocol instances.

Total: The total value does not represent the sum of all profile tasks, but the maximum value allowed per CPU. Therefore, the user can create several task types, while the established numbers for each one and the total value are not surpassed.

4.3. CPU Configuration

The Nexto CPU configuration is located in the device tree, as shown on figure below, and can be accessed by a double-click on the corresponding object. In this tab it's possible to configure the diagnostics area, the retentive and persistent memory area and hot swap mode, among other parameters, as described in the CPU Configuration.

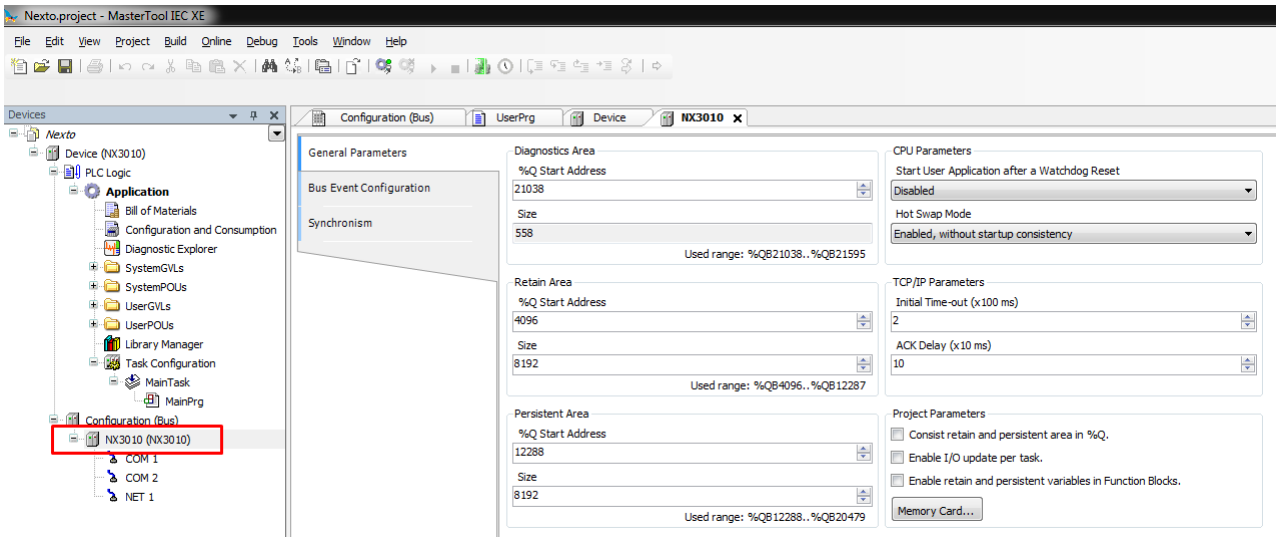


Figure 19: CPU Configuration

Besides that, by double-clicking on CPU's NET 1 icon, it's possible to configure the Ethernet interface that will be used for communication between the controller and the software MasterTool IEC XE.

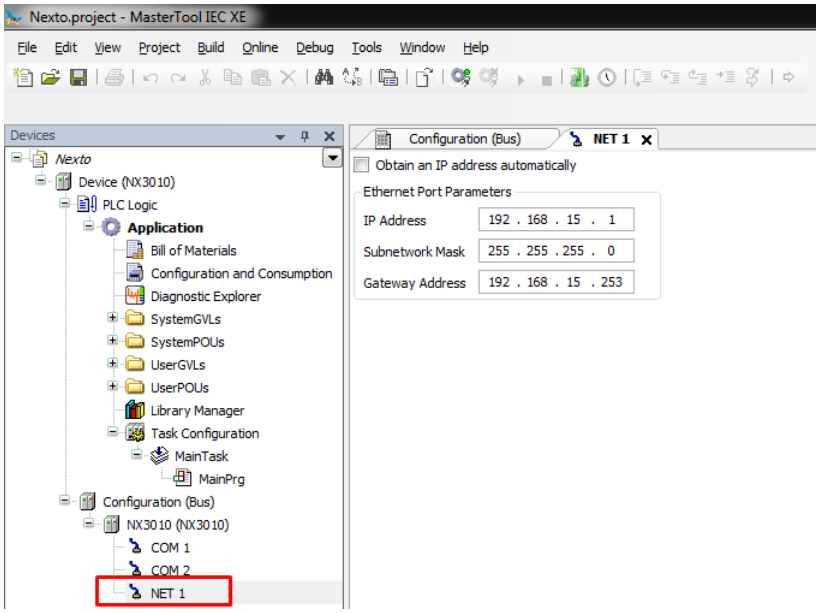


Figure 20: Configuring the CPU Communication Port

The configuration defined on this tab will be applied to the device only when sending the application to the device (download), which is described further on sections Finding the Device and Login.

4.4. Libraries

There are several programming tool resources which are available through libraries. Therefore, these libraries must be inserted in the project so its utilization becomes possible. The insertion procedure and more information about available libraries must be found in the MasterTool Programming Manual – MP399609.

4.5. Inserting a Protocol Instance

The Nexto Series CPUs, as described in the Protocols section, offers several communication protocols. Except for the OPC DA and OPC UA communication, which have a different configuration procedure, the insertion of a protocol can be done by simply right-clicking on the desired communication interface, selecting to add the device and finally performing the configuration as shown in the Protocols Configuration section. Below is presented an examples.

4.5.1. MODBUS Ethernet

The first step to configure the MODBUS Ethernet (Client in this example), is to include the instance in the desired NET (in this case, NET 1, as the CPU NX3010 has only one Ethernet interface). Click on the *NET* with the mouse right button and select *Add Device...*, as shown on figure below.

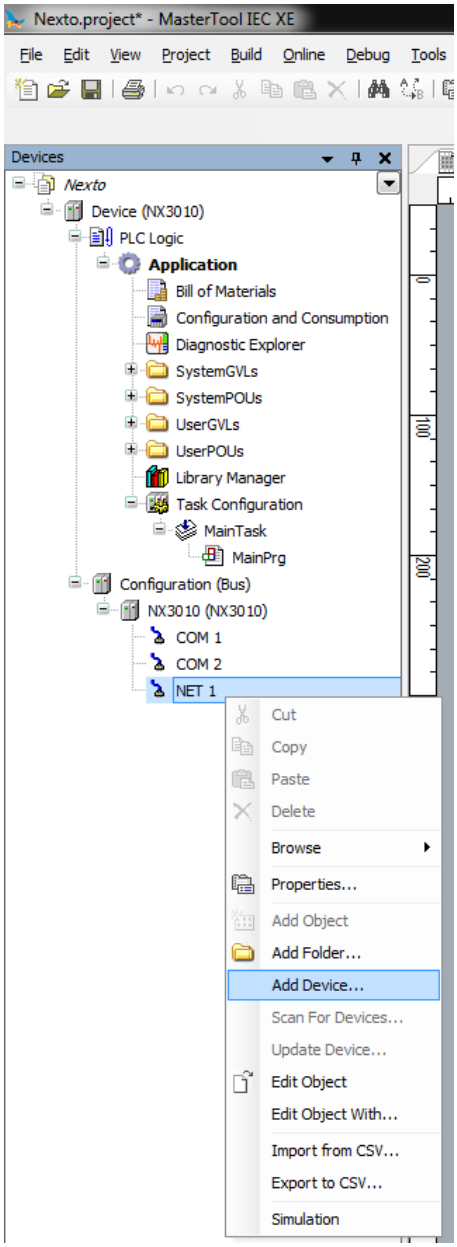


Figure 21: Adding the Instance

After that, the available protocols for the user will appear on the screen. In this menu is defined the configuration mode of the protocol. Selecting the option *MODBUS Symbol Client*, for Symbolic Mapping setting or *MODBUS Client*, for Direct Addressing (%Q). Then, click *Add Device*, as shown in the figure below.

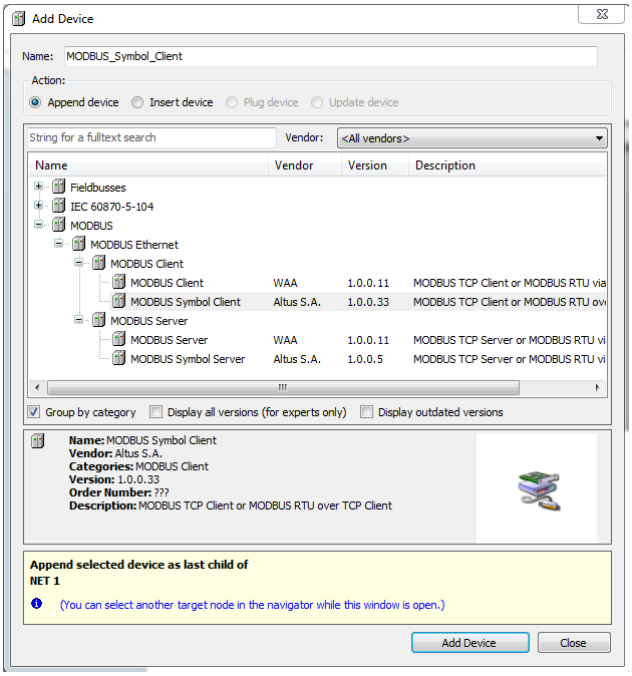


Figure 22: Selecting the Protocol

4.6. Finding the Device

To establish the communication between the CPU and MasterTool IEC XE, first it's necessary to find and select the desired device. The configuration of this communication is located on the object *Device* on device tree, on *Communication Settings* tab. On this tab, after selecting the *Gateway* and clicking on button *Scan network*, the software MasterTool IEC XE performs a search for devices and shows the CPUs found on the network of the Ethernet interface of the station where the tool is running.

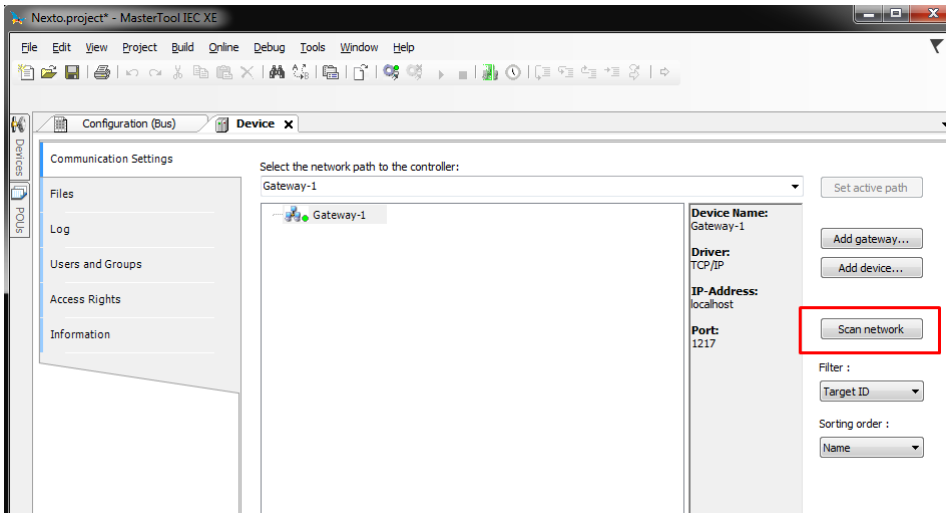


Figure 23: Finding the CPU

If there is no gateway previously configured, it can be included by the button *Add gateway*, using the default IP address *localhost* to use the gateway resident on the station or changing the IP address to use the device internal gateway.

Next, the desired controller must be selected by clicking on *Set active path*. This action selects the controller and informs the configuration software which controller shall be used to communicate and send the project.

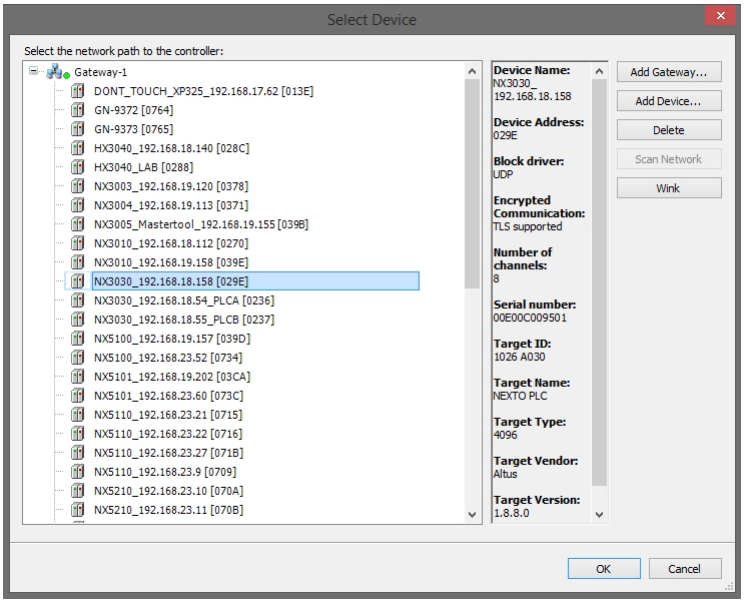


Figure 24: Selecting the CPU

Additionally, the user can change the default name of the device that is displayed. For that, you must click the right mouse button on the desired device and select *Change Device Name*. After a name change, the device will not return to the default name under any circumstances.

In case the Ethernet configuration of the CPU to be connected is in a different network from the Ethernet interface of the station, the software MasterTool IEC XE will not be able to find the device. In this case, it's recommended to use the command *Easy Connection* located on Online menu.

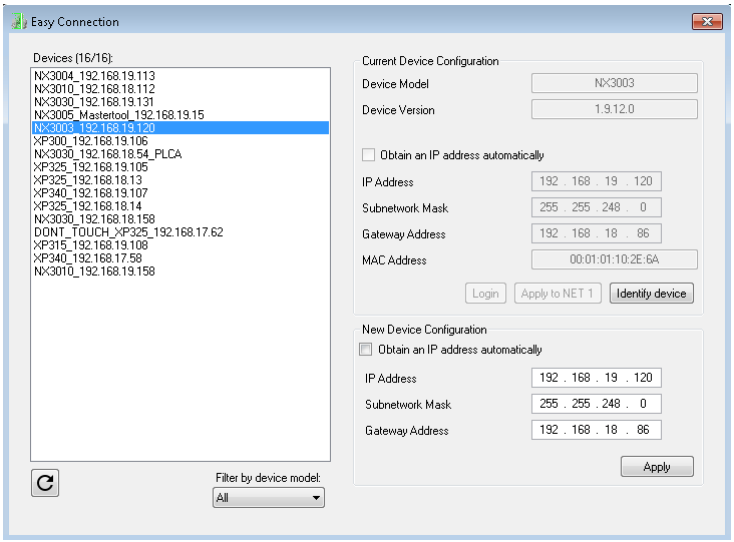


Figure 25: Easy Connection

This command performs a MAC level communication with the NET 1 interface of the device, allowing to permanently change the configuration of the CPU's Ethernet interface, independently of the IP configuration of the station and from the one previously configured on the device. So, with this command, it's possible to change the device configuration to put it on the same network of the Ethernet interface of the station where MasterTool IEC XE is running, allowing to find and select the device for the communication. The complete description of *Easy Connection* command can be found on User Manual of MasterTool IEC XE code MU299609.

4.7. Login

After compiling the application and fixing errors that might be found, it's time to send the project to the CPU. To do this, simply click on *Login* command located on *Online* menu of MasterTool IEC XE as shown on the following figure. This operation may take a few seconds, depending on the size of the generated file.

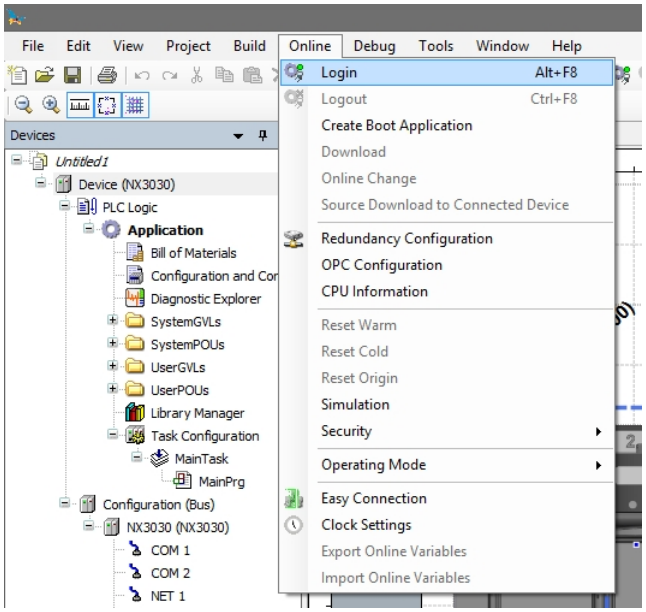


Figure 26: Sending the Project to the CPU

After the command execution, some user interface messages may appear, which are presented due to differences between an old project and the new project been sent, or simply because there was a variation in some variable.

If the Ethernet configuration of the project is different from the device, the communication may be interrupted at the end of download process when the new configuration is applied on the device. So, the following warning message will be presented, asking the user to proceed or not with this operation.

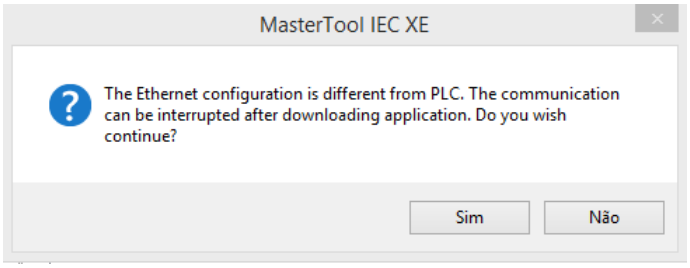


Figure 27: IP Configuration Warning

If there is no application on the CPU, the following message will be presented.

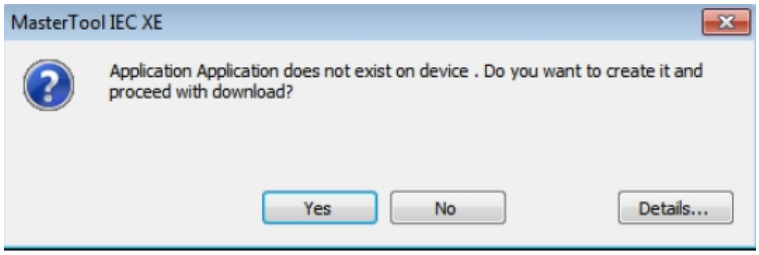


Figure 28: No application on the device

If there is already an application on the CPU, depending on the differences between the projects, the following options will be presented:

- **Login with online change:** execute the login and send the new project without stopping the current CPU application (see Run Mode item), updating the changes when a new cycle is executed.
- **Login with download:** execute the login and send the new project with the CPU stopped (see Stop Mode). When the application is initiated, the update will have been done already.
- **Login without any change:** executes the login without sending the new project.

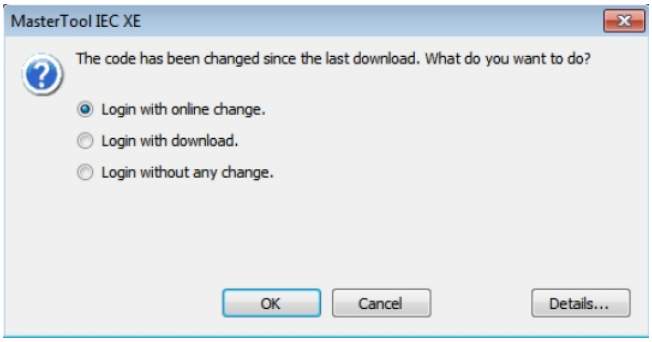


Figure 29: Project Update at the CPU

ATTENTION

In the online changes is not permitted to associate symbolic variables mapping from a global variable list (GVL) and use these variables in another global variable list (GVL).

If the new application contains changes on the configuration, the online change will not be possible. In this case, the MasterTool IEC XE requests whether the login must be executed as download (stopping the application) or if the operation must be cancelled.

PS.: The button *Details...* shows the changes made in the application.

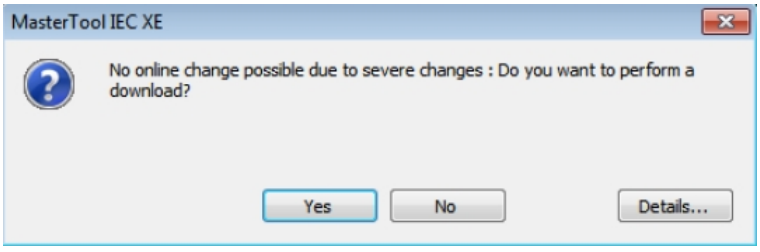


Figure 30: Configuration Changes

Finally, at the end of this process the MasterTool IEC XE offers the option to transfer (download) the source code to the internal memory of the device, as shown on the following figure:

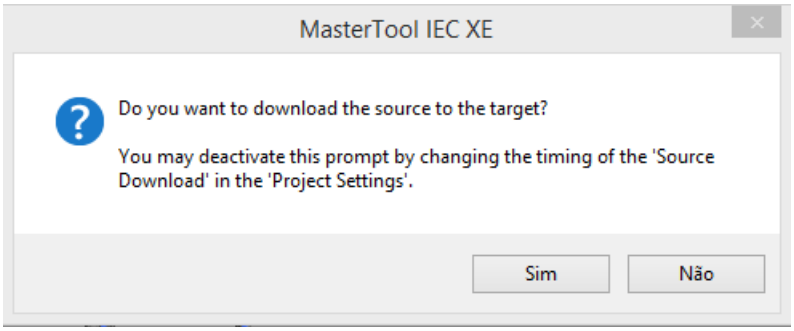


Figure 31: Source code download

Transferring the source code is fundamental to ensure the future restoration of the project and to perform modifications on the application that is loaded into the device.

4.8. Run Mode

Right after the project has been sent to the CPU, the application will not be immediately executed (except for the case of an online change). For that to happen, the command Start must be executed. This way, the user can control the execution of the application sent to the CPU, allowing pre-configuring initial values which will be used by the CPU on the first execution cycle.

To execute this command, simply go to the *Debug* menu and select the option *Start*, as shown on figure below.

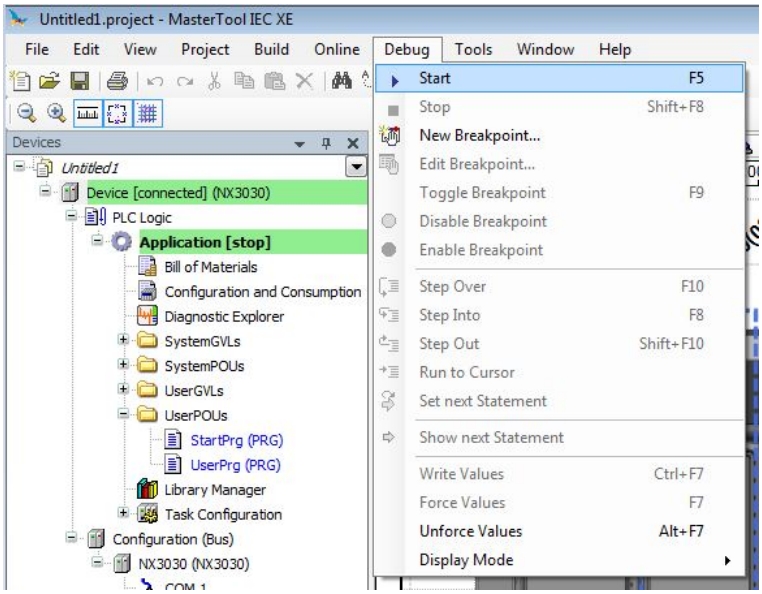


Figure 32: Starting the Application

The figure below shows the application in execution. In case the POU tab is selected, the created variables are listed on a monitoring window, in which the values can be visualized and forced by the user.

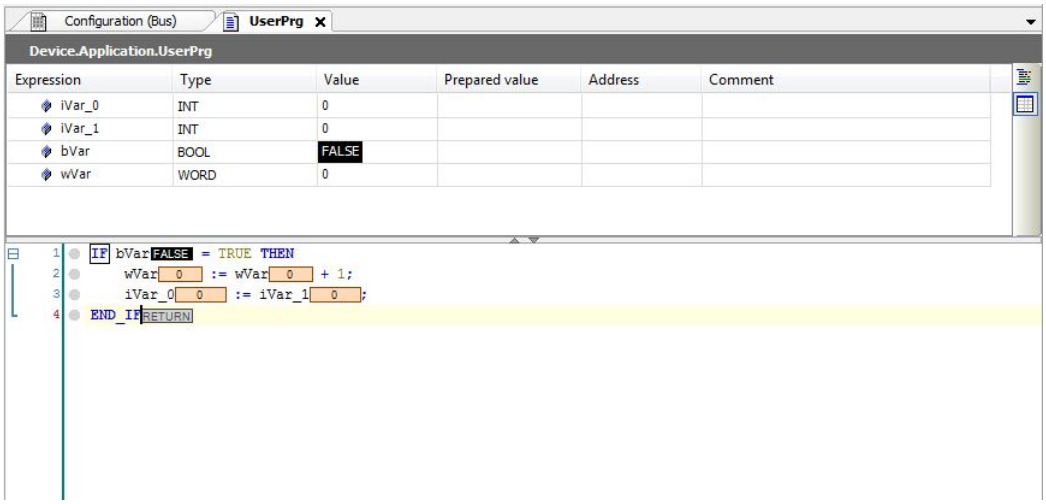


Figure 33: Program running

If the CPU already have a boot application internally stored, it goes automatically to Run Mode when the device is powered on, with no need for an online command through MasterTool IEC XE.

4.9. Stop Mode

To stop the execution of the application, the user must execute the *Stop* command, available at the menu *Debug*, as shown on figure below.

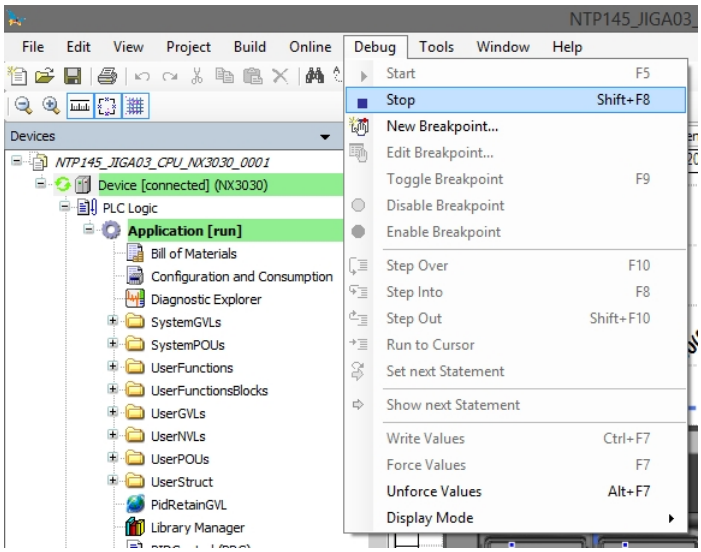


Figure 34: Stopping the Application

In case the CPU is initialized without the stored application, it automatically goes to Stop Mode, as it happens when a software exception occurs.

4.10. Writing and Forcing Variables

After Logging into a PLC, the user can write or force values to a variable of the project.

The writing command (*CTRL + F7*) writes a value into a variable and this value could be overwritten by instructions executed in the application.

Moreover, the forced writing command (*F7*) writes a value into a variable without allowing this value to be changed until the forced variables are released.

It is important to highlight that, when used the MODBUS RTU Slave and the MODBUS Ethernet Server, and the *Read-only* option from the configured relations is not selected, the forced writing command (*F7*) must be done over the available variables in the monitoring window as the writing command (*CTRL + F7*) leaves the variables to be overwritten when new readings are done.

ATTENTION

The variables forcing can be done in Online mode.
Diagnostic variables cannot be forced, only written, because diagnostics are provided by the CPU and will be overwritten by it.

When a forced writing is performed in a redundant variable of the Active PLC, the application's MainTask will suffer an impact on its execution time, both in the Active PLC and in the Reserve PLC. This is because the two Half-Clusters will exchange information about the forced variables every cycle. Therefore, when forcing variables in a redundant system, the addition that the task execution can have must be considered. The table below exemplifies how much will be increased, on average, in the execution of MainTask when this occurs:

	Active CP			Reserve CP		
Runtime	**50 ms**	**100 ms**	**200 ms**	**50 ms**	**100 ms**	**200 ms**
Addition with 10 forces	2,4 %	2,2 %	1,7 %	4,0 %	3,4 %	2,0 %
Add to 50 forces	12,0 %	9,2 %	6,0 %	18,0 %	12,0 %	8,0 %
Add with 128 forcings	26,0 %	21,0 %	16,0 %	56,0 %	34,0 %	22,5 %

Table 42: Influence of Forcing Variables on a Redundant CP

ATTENTION

When a CPU is with forced variables and it is de-energized, the variables will lose the forcing in the next initialization.
The limit of forcing for the Nexto CPUs is 128 variables, regardless of model or configuration of CPU used.

4.11. Logout

To finalize the online communication with the CPU, the command *Logout* must be executed, located in the *Online* menu, as shown on figure below.

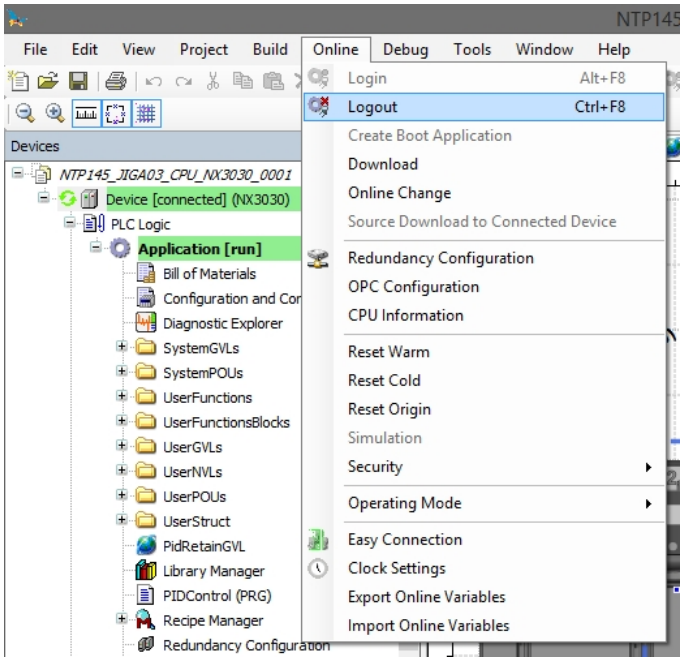


Figure 35: Finalizing the online communication with the CPU

4.12. Project Upload

Nexto Series CPUs are capable to store the source code of the application on the internal memory of the device, allowing future retrieval (upload) of the complete project and to modify the application.

To recover a project previously stored on the internal memory of the CPU, the command located on *File* menu must be executed as shown on the following figure.

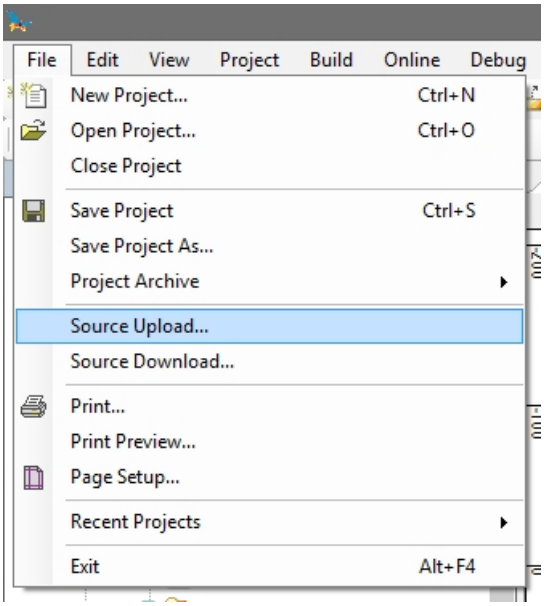


Figure 36: Project Upload Option

Next, just select the desired CPU and click *OK*, as shown on figure below.

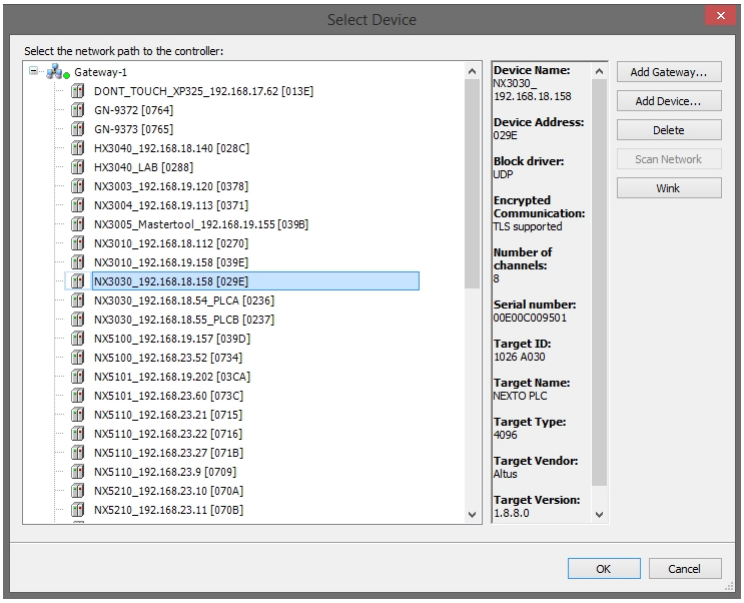


Figure 37: Selecting the CPU

To ensure that the project loaded in the CPU is identical and can be accessed in other workstations, consult the chapter *Projects Download/Login Method without Project Differences* at the MasterTool IEC XE User Manual MT8500 - MU299609.

ATTENTION

The memory size area to store a project in the Nexto CPUs is defined on section Memory.

ATTENTION

The upload recovers the last project stored in the controller as described in the previous paragraphs. In case only the application was downloaded, without transferring its source code, it will not be possible to be recovered by the Upload procedure.

4.13. CPU Operating States

4.13.1. Run

When a CPU is in *Run* mode, all application tasks are executed.

4.13.2. Stop

When a CPU is in *Stop* mode, all application tasks are stopped. The variable values in the tasks are kept with the current value and output points go to the safe state.

When a CPU goes to the *Stop* mode due to the download of an application, the variables in the application tasks will be lost except the persistent variables type.

4.13.3. Breakpoint

When a debugging mark is reached in a task, it is interrupted. All the active tasks in the application will not be interrupted, continuing their execution. With this feature, it's possible to go through and investigate the program flow step by step in *Online* mode according to the positions of the interruptions included through the editor.

For further information about the use of breakpoints, please consult the MasterTool IEC XE Utilization Manual - MU299609.

4.13.4. Exception

When a CPU is in *Exception* it indicates that some improper operation occurred in one of the application active tasks. The task which caused the Exception will be suspended and the other tasks will pass for the Stop mode. It is only possible to take off the tasks from this state and set them in execution again after a new CPU start condition. Therefore, only with a *Reset Warm*, *Reset Cold*, *Reset Origin* or a CPU restart puts the application again in Run mode.

4.13.5. Reset Warm

This command puts the CPU in *Stop* mode and initializes all the application tasks variables, except the persistent and retentive type variables. The variables initialized with a specific value will assume exactly this value, the other variables will assume the standard initialization value (zero).

4.13.6. Reset Cold

This command puts the CPU in *Stop* mode and initializes all the application tasks variables, except the persistent type variables. The variables initialized with a specific value will assume exactly this value, the other variables will assume the standard initialization value (zero).

4.13.7. Reset Origen

This command removes all application task variables, including persistent type variables, deletes the application CPU and puts the CPU in Stop mode.

Notes:

Reset: When a Reset is executed, the breakpoints defined in the application are disabled.

Command: To execute the commands *Reset Warm*, *Cold* or *Origin*, it's necessary to have MasterTool IEC XE in *Online* mode with the CPU.

4.13.8. Reset Process Command (IEC 60870-5-104)

This process reset command can be solicited by IEC 60870-5-104 clients. After answer the client, the CPU start a rebooting process, as if being done an energizing cycle.

In case of redundant PLCs, the process reset command is synchronized with the non-active PLC, resulting the reboot of both PLCs.

The standard IEC 60870-5-104 foresee a qualification value pass (0..255) with the process reset command, but this "parameter" is not considered by the CPU.

4.14. Programs (POUs) and Global Variable Lists (GVLs)

The project created by MasterTool IEC XE contains a set of program modules (POUs) and global variables lists that aims to facilitate the programming and utilization of the controller. The following sections describe the main elements that are part of this standard project structure.

4.14.1. MainPrg Program

The MainTask is associated to one unique POU of program type, named MainPrg. The MainPrg program is created automatically and cannot be edited by user.

The MainPrg program code is the following, in ST language:

```
(*Main POU associated with MainTask that calls StartPrg,
 UserPrg/ActivePrg and NonSkippedPrg.
 This POU is blocked to edit.*)

PROGRAM MainPrg
VAR
   isFirstCycle : BOOL := TRUE;
END_VAR
```

```
SpecialVariablesPrg();
IF isFirstCycle THEN
   StartPrg();
   isFirstCycle := FALSE;
ELSE
   UserPrg();
END_IF;
```

MainPrg call other two POUs of program type, named *StartPrg* and *UserPrg*. While the *UserPrg* is always called, the *StartPrg* is only called once in the PLC application start.

To the opposite of *MainPrg* program, that must not be modified, the user can change the *StartPrg* and *UserPrg* programs. Initially, when the project is created from the wizard, these two programs are created *empty*, but the user might insert code in them.

4.14.2. StartPrg Program

In this POU the user might create logics, loops, start variables, etc. that will be executed only one time in the first PLC's cycle, before execute *UserPrg* POU by the first time. And not being called again during the project execution.

In case the user load a new application, or if the PLC gets powered off, as well as in *Reset Origin*, *Reset Cold* and *Reset Warm* conditions, this POU is going to be executed again.

4.14.3. UserPrg Program

In this POU the user must create the main application, responsible by its own process control. This POU is called by the main POU (MainPrg).

The user can also create additional POUs (programs, functions or function blocks), and called them or instance them inside UserPrg POU, to ends of its program instruction. Also it is possible to call functions and instance function blocks defined in libraries.

4.14.4. GVL System_Diagnostics

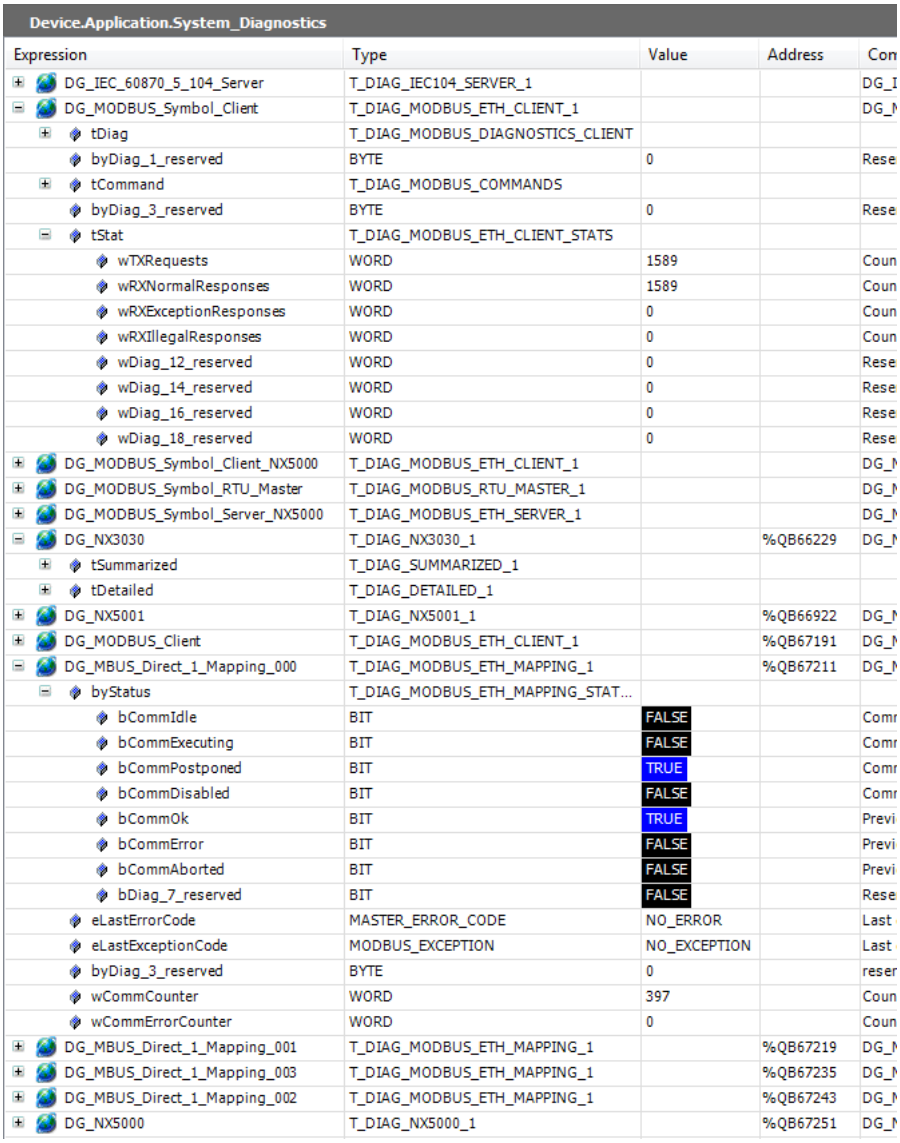
The *System_Diagnostics* GVL contains the diagnostic variables of the CPU, of the communication interface (Ethernet and PROFIBUS) and of all communication drivers. This GVL isn't editable and the variables are declared automatically with type specified by the device to which it belongs when it is added to the project.

ATTENTION

In *System_Diagnostics* GVL, are also declared the diagnostic variables of the direct representation MODBUS Client/Master Requests.

Some devices, like the MODBUS Symbol communication driver, doesn't have its diagnostics allocated at %Q variables with the AT directive. The same occurs with newest communication drivers, as Server IEC 60870-5-104.

The following picture shows an example of the presentation of this GVL when in *Online* mode.



Device.Application.System_Diagnostics

Expression	Type	Value	Address	Con
DG_IEC_60870_5_104_Server	T_DIAG_IEC104_SERVER_1			DG_I
DG_MODBUS_Symbol_Client	T_DIAG_MODBUS_ETH_CLIENT_1			DG_M
tDiag	T_DIAG_MODBUS_DIAGNOSTICS_CLIENT			
byDiag_1_reserved	BYTE	0		Rese
tCommand	T_DIAG_MODBUS_COMMANDS			
byDiag_3_reserved	BYTE	0		Rese
tStat	T_DIAG_MODBUS_ETH_CLIENT_STATS			
wTXRequests	WORD	1589		Coun
wRXNormalResponses	WORD	1589		Coun
wRXExceptionResponses	WORD	0		Coun
wRXIllegalResponses	WORD	0		Coun
wDiag_12_reserved	WORD	0		Rese
wDiag_14_reserved	WORD	0		Rese
wDiag_16_reserved	WORD	0		Rese
wDiag_18_reserved	WORD	0		Rese
DG_MODBUS_Symbol_Client_NX5000	T_DIAG_MODBUS_ETH_CLIENT_1			DG_M
DG_MODBUS_Symbol_RTU_Master	T_DIAG_MODBUS_RTU_MASTER_1			DG_M
DG_MODBUS_Symbol_Server_NX5000	T_DIAG_MODBUS_ETH_SERVER_1			DG_M
DG_NX3030	T_DIAG_NX3030_1		%QB66229	DG_M
tSummarized	T_DIAG_SUMMARIZED_1			
tDetailed	T_DIAG_DETAILED_1			
DG_NX5001	T_DIAG_NX5001_1		%QB66922	DG_M
DG_MODBUS_Client	T_DIAG_MODBUS_ETH_CLIENT_1		%QB67191	DG_M
DG_MBUS_Direct_1_Mapping_000	T_DIAG_MODBUS_ETH_MAPPING_1		%QB67211	DG_M
byStatus	T_DIAG_MODBUS_ETH_MAPPING_STAT...			
bCommIdle	BIT	FALSE		Comm
bCommExecuting	BIT	FALSE		Comm
bCommPostponed	BIT	TRUE		Comm
bCommDisabled	BIT	FALSE		Comm
bCommOk	BIT	TRUE		Previ
bCommError	BIT	FALSE		Previ
bCommAborted	BIT	FALSE		Previ
bDiag_7_reserved	BIT	FALSE		Rese
eLastErrorCode	MASTER_ERROR_CODE	NO_ERROR		Last
eLastExceptionCode	MODBUS_EXCEPTION	NO_EXCEPTION		Last
byDiag_3_reserved	BYTE	0		reser
wCommCounter	WORD	397		Coun
wCommErrorCounter	WORD	0		Coun
DG_MBUS_Direct_1_Mapping_001	T_DIAG_MODBUS_ETH_MAPPING_1		%QB67219	DG_M
DG_MBUS_Direct_1_Mapping_003	T_DIAG_MODBUS_ETH_MAPPING_1		%QB67235	DG_M
DG_MBUS_Direct_1_Mapping_002	T_DIAG_MODBUS_ETH_MAPPING_1		%QB67243	DG_M
DG_NX5000	T_DIAG_NX5000_1		%QB67251	DG_M

Figure 38: System_Diagnostics GVL in Online Mode

4.14.5. GVL Disables

The *Disables* GVL contains the MODBUS Master/Client by symbolic mapping requisition disabling variables. It is not mandatory, but it is recommended to use the automatic generation of these variables, which is done clicking in the button *Generate Disabling Variables* in device requisition tab. These variables are declared as type BOOL and follow the following structure:

Requisition disabling variables declaration:

```
[Device Name]_DISABLE_[Requisition Number] : BOOL;
```

Where:

Device name: Name that shows on Tree View to the MODBUS device.

Requisition Number: Requisition number that was declared on the MODBUS device requisition table following the sequence from up to down, starting on 0001.

Example:

Device.Application.Disables

```
VAR_GLOBAL
  MODBUS_Device_DISABLE_0001   : BOOL;
  MODBUS_Device_DISABLE_0002   : BOOL;
  MODBUS_Device_DISABLE_0003   : BOOL;
  MODBUS_Device_1_DISABLE_0001   : BOOL;
  MODBUS_Device_1_DISABLE_0002   : BOOL;
END_VAR
```

The automatic generation through button *Generate Disabling Variables* only create variables, and don't remove automatically. This way, in case any relation is removed, its respective disabling variable must be removed manually.

The *Disables* GVL is editable, therefore the requisition disabling variables can be created manually without need of following the model created by the automatic declaration and can be used both ways at same time, but must always be of BOOL type. And it is need to take care to do not delete or change the automatic declared variables, cause them can being used for some MODBUS device. If the variable be deleted or changed then an error is going to be generated while the project is being compiled. To correct the automatically declared variable name, it must be followed the model exemplified above according to the device and the requisition to which they belong.

The following picture shows an example of the presentation of this GVL when in *Online* mode. If the variable values are TRUE it means that the requisition to which the variables belong is disabled and the opposite is valid when the variable value is FALSE.

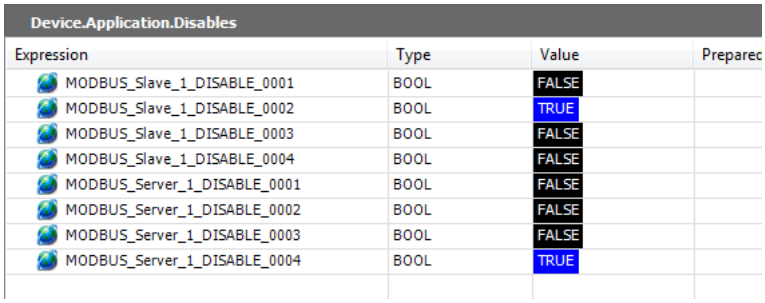


Figure 39: Disable GVL in Online Mode

4.14.6. GVL IOQualities

The *IOQualities* GVL contains the quality variables of I/O modules declared on CPU's bus. This GVL is not editable and the variables are automatically declared as *LibDataTypes.QUALITY* type arrays, and dimensions according to I/Os quantities of the module to which it belongs when that is added to the project.

Example: Device.Application.IOQualities

```
VAR_GLOBAL
  QUALITY_NX1001: ARRAY[0..15]  OF LibDataTypes.QUALITY;
  QUALITY_NX2020: ARRAY[0..15]  OF LibDataTypes.QUALITY;
  QUALITY_NX6000: ARRAY[0..7]   OF LibDataTypes.QUALITY;
  QUALITY_NX6100: ARRAY[0..3]   OF LibDataTypes.QUALITY;
END_VAR
```

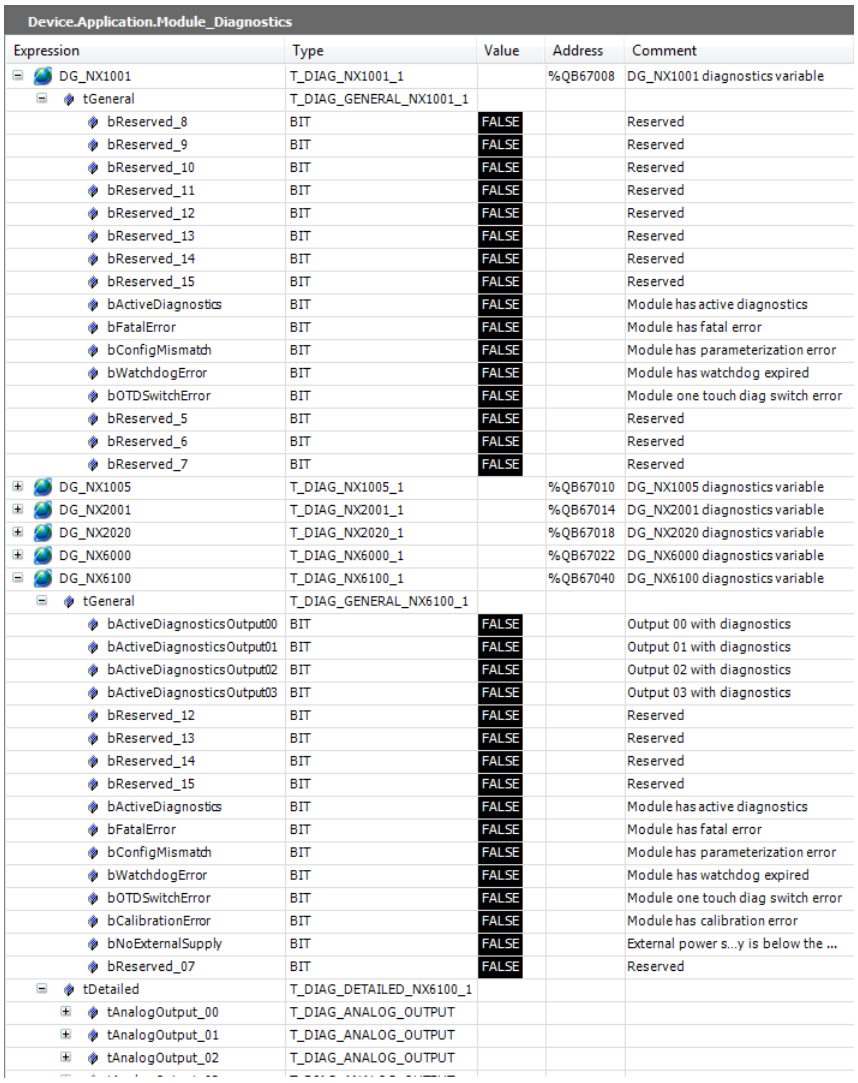
Once the application is in *RUN* it is possible to watch the I/O modules quality variables values that were added to the project through *IOQualities* GVL.

4.14.7. GVL Module_Diagnostics

The *Module_Diagnostics* GVL contains the diagnostics variables of the I/O modules used in the project, except by the CPU and communication drivers. This GVL isn't editable and the variables are automatically declared with type specified by

the module, to which it belongs, when that is added to the project.

The following picture shows an example of the presentation of this GVL when in *Online* mode.



Device.Application.Module_Diagnostics

Expression	Type	Value	Address	Comment
DG_NX1001	T_DIAG_NX1001_1		%QB67008	DG_NX1001 diagnostics variable
tGeneral	T_DIAG_GENERAL_NX1001_1			
bReserved_8	BIT	FALSE		Reserved
bReserved_9	BIT	FALSE		Reserved
bReserved_10	BIT	FALSE		Reserved
bReserved_11	BIT	FALSE		Reserved
bReserved_12	BIT	FALSE		Reserved
bReserved_13	BIT	FALSE		Reserved
bReserved_14	BIT	FALSE		Reserved
bReserved_15	BIT	FALSE		Reserved
bActiveDiagnostics	BIT	FALSE		Module has active diagnostics
bFatalError	BIT	FALSE		Module has fatal error
bConfigMismatch	BIT	FALSE		Module has parameterization error
bWatchdogError	BIT	FALSE		Module has watchdog expired
bOTDSwitchError	BIT	FALSE		Module one touch diag switch error
bReserved_5	BIT	FALSE		Reserved
bReserved_6	BIT	FALSE		Reserved
bReserved_7	BIT	FALSE		Reserved
DG_NX1005	T_DIAG_NX1005_1		%QB67010	DG_NX1005 diagnostics variable
DG_NX2001	T_DIAG_NX2001_1		%QB67014	DG_NX2001 diagnostics variable
DG_NX2020	T_DIAG_NX2020_1		%QB67018	DG_NX2020 diagnostics variable
DG_NX6000	T_DIAG_NX6000_1		%QB67022	DG_NX6000 diagnostics variable
DG_NX6100	T_DIAG_NX6100_1		%QB67040	DG_NX6100 diagnostics variable
tGeneral	T_DIAG_GENERAL_NX6100_1			
bActiveDiagnosticsOutput00	BIT	FALSE		Output 00 with diagnostics
bActiveDiagnosticsOutput01	BIT	FALSE		Output 01 with diagnostics
bActiveDiagnosticsOutput02	BIT	FALSE		Output 02 with diagnostics
bActiveDiagnosticsOutput03	BIT	FALSE		Output 03 with diagnostics
bReserved_12	BIT	FALSE		Reserved
bReserved_13	BIT	FALSE		Reserved
bReserved_14	BIT	FALSE		Reserved
bReserved_15	BIT	FALSE		Reserved
bActiveDiagnostics	BIT	FALSE		Module has active diagnostics
bFatalError	BIT	FALSE		Module has fatal error
bConfigMismatch	BIT	FALSE		Module has parameterization error
bWatchdogError	BIT	FALSE		Module has watchdog expired
bOTDSwitchError	BIT	FALSE		Module one touch diag switch error
bCalibrationError	BIT	FALSE		Module has calibration error
bNoExternalSupply	BIT	FALSE		External power s...y is below the ...
bReserved_07	BIT	FALSE		Reserved
tDetailed	T_DIAG_DETAILED_NX6100_1			
tAnalogOutput_00	T_DIAG_ANALOG_OUTPUT			
tAnalogOutput_01	T_DIAG_ANALOG_OUTPUT			
tAnalogOutput_02	T_DIAG_ANALOG_OUTPUT			

Figure 40: Module_Diagnostics GVL in Online Mode

4.14.8. GVL Qualities

The *Qualities* GVL contains the quality variable of the internal variables MODBUS Master/Client of symbolic mapping. It is not mandatory but is recommended to use these variables' automatic generation, what is done clicking on button *Generate Quality Variables* in the device mapping tab. These variables are declared as *LibDataTypes.QUALITY* type and follow the following structure:

Quality mapping variable declaration:

```
[Device Name]_QUALITY_[Mapping Number]: LibDataTypes.QUALITY;
```

Where:

Device Name: Name that appear at the Tree View to the device.

Mapping Number: Number of the mapping that was declared on the device mapping table, following the up to down sequence, starting with 0001.

ATTENTION

It is not possible to associate quality variables to the direct representation MODBUS Master/Client drivers' mappings. Therefore it is recommended the use of symbolic mapping MODBUS drivers.

Example: Device.Application.Qualities

```
VAR_GLOBAL
  MODBUS_Device_QUALITY_0001: LibDataTypes.QUALITY;
  MODBUS_Device_QUALITY_0002: LibDataTypes.QUALITY;
  MODBUS_Device_QUALITY_0003: LibDataTypes.QUALITY;
END_VAR
```

The *Qualities* GVL is editable, therefore the mapping quality variables can be created manually without need to follow the automatic declaration model, and can be used both ways at same time. But must always be of *LibDataTypes.QUALITY* type and take care to don't delete or change a variable automatically declared, because they might being used by some device. If the variable be deleted or changed an error is going to be generated while the project is being compiled. To correct the automatically declared variable name, it must be followed the model exemplified above according to the device and the requisition to which they belong.

To the MODBUS communication devices the quality variables behave on the way showed at Table 56.

The following picture shows an example of the presentation of this GVL when in *Online* mode.

ATTENTION

If a symbolic mapping MODBUS Client/Master driver's variable be mapped in IEC 60870-5-104 Server driver, it is necessary that the MODBUS mapping quality variables had been created to generate valid quality events to such IEC 60870-5-104 Server points. Case opposite, aren't going to be generated "bad" quality events to IEC 60870-5-104 Server clients in the situations that MODBUS Master/Client can't communicate with its slaves/servers, by example.

Device.Application.Qualities

Expression	Type	Value	Address	Comment
MODBUS_Slave_1_QUALITY_0001	LibDataTypes.QUALITY			
VALIDITY	QUALITY_VALIDITY	VALIDITY_GOOD		Quality validity
FLAGS	QUALITY_FLAGS			Quality flags
FLAG_OUT_OF_RANGE	BIT	FALSE		Bit 8
FLAG_INACCURATE	BIT	FALSE		Bit 9
FLAG_OLD_DATA	BIT	FALSE		Bit 10
FLAG_FAILURE	BIT	FALSE		Bit 11
FLAG_OPERATOR_BLOCKED	BIT	FALSE		Bit 12
FLAG_TEST	BIT	FALSE		Bit 13
FLAG_RESERVED_0	BIT	FALSE		Bit 14
FLAG_RESERVED_1	BIT	FALSE		Bit 15
FLAG_RESTART	BIT	FALSE		Bit 0
FLAG_COMM_FAIL	BIT	FALSE		Bit 1
FLAG_REMOTE_SUBSTITU...	BIT	FALSE		Bit 2
FLAG_LOCAL_SUBSTITUTED	BIT	FALSE		Bit 3
FLAG_FILTER	BIT	FALSE		Bit 4
FLAG_OVERFLOW	BIT	FALSE		Bit 5
FLAG_REFERENCE_ERROR	BIT	FALSE		Bit 6
FLAG_INCONSISTENT	BIT	FALSE		Bit 7
MODBUS_Slave_1_QUALITY_0002	LibDataTypes.QUALITY			
MODBUS_Slave_1_QUALITY_0003	LibDataTypes.QUALITY			
MODBUS_Slave_1_QUALITY_0004	LibDataTypes.QUALITY			
MODBUS_Server_1_QUALITY_0001	LibDataTypes.QUALITY			
MODBUS_Server_1_QUALITY_0002	LibDataTypes.QUALITY			
MODBUS_Server_1_QUALITY_0003	LibDataTypes.QUALITY			
VALIDITY	QUALITY_VALIDITY	VALIDITY_QUESTIONABLE		Quality validity
FLAGS	QUALITY_FLAGS			Quality flags
FLAG_OUT_OF_RANGE	BIT	FALSE		Bit 8
FLAG_INACCURATE	BIT	FALSE		Bit 9
FLAG_OLD_DATA	BIT	TRUE		Bit 10
FLAG_FAILURE	BIT	FALSE		Bit 11
FLAG_OPERATOR_BLOCKED	BIT	FALSE		Bit 12
FLAG_TEST	BIT	FALSE		Bit 13
FLAG_RESERVED_0	BIT	FALSE		Bit 14
FLAG_RESERVED_1	BIT	FALSE		Bit 15
FLAG_RESTART	BIT	FALSE		Bit 0
FLAG_COMM_FAIL	BIT	TRUE		Bit 1
FLAG_REMOTE_SUBSTITU...	BIT	FALSE		Bit 2
FLAG_LOCAL_SUBSTITUTED	BIT	FALSE		Bit 3
FLAG_FILTER	BIT	FALSE		Bit 4
FLAG_OVERFLOW	BIT	FALSE		Bit 5
FLAG_REFERENCE_ERROR	BIT	FALSE		Bit 6
FLAG_INCONSISTENT	BIT	FALSE		Bit 7
MODBUS_Server_1_QUALITY_0004	LibDataTypes.QUALITY			

Figure 41: Qualities GVL in Online Mode

4.14.9. GVL ReqDiagnostics

The *ReqDiagnostics* GVL contains the requisition diagnostics variables of symbolic mapping MODBUS Master/Client. It is not mandatory, but recommended the use of these variables' automatic generation, what is done by clicking in the button *Generate Diagnostics Variables* in device requests tab. These variables declaration follow the following structure:

Requisition diagnostic variable declaration:

```
[Device Name]_REQDG_[Requisition Number]: [Variable Type];
```

Where:

Device Name: Name that appear at the Tree View to the device.

Mapping Number: Number of the mapping that was declared on the device mapping table, following the up to down sequence, starting with 0001.

Variable Type: NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS_RTU_MAPPING_1 to MODBUS Master and NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS_ETH_MAPPING_1 to MODBUS Client.

ATTENTION

The requisition diagnostics variables of direct mapping MODBUS Master/Client are declared at *System_Diagnostics* GVL.

Example:

Device.Application.ReqDiagnostics

```
VAR_GLOBAL
  MODBUS_Device_REQDG_0001   : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_RTU_MAPPING_1;
  MODBUS_Device_REQDG_0002   : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_RTU_MAPPING_1;
  MODBUS_Device_REQDG_0003   : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_RTU_MAPPING_1;
  MODBUS_Device_1_REQDG_0001  : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_ETH_MAPPING_1;
  MODBUS_Device_1_REQDG_0002  : NXMODBUS_DIAGNOSTIC_STRUCTS.
                               T_DIAG_MODBUS_ETH_MAPPING_1;
END_VAR
```

The *ReqDiagnostics* GVL is editable, therefore the requisitions diagnostic variables can be manually created without need to follow the model created by the automatic declaration. Both ways can be used at same time, but the variables must always be of type referring to the device. And take care to don't delete or change a variable automatically declared, because they might being used by some device. If the variable be deleted or changed an error is going to be generated while the project is being compiled. To correct the automatically declared variable name, it must be followed the model exemplified above according to the device and the requisition to which they belong.

The following picture shows an example of the presentation of this GVL when in *Online* mode.

Device.Application.ReqDiagnostics

Expression	Type	Value
MODBUS_Slave_1_REQDG_0001	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
byStatus	T_DIAG_MODBUS_RTU_MAPPING_STATUS	
eLastErrorCode	MASTER_ERROR_CODE	NO_ERROR
eLastExceptionCode	MODBUS_EXCEPTION	NO_EXCEPTION
byDiag_3_reserved	BYTE	0
wCommCounter	WORD	969
wCommErrorCounter	WORD	0
MODBUS_Slave_1_REQDG_0002	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
MODBUS_Slave_1_REQDG_0003	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
MODBUS_Slave_1_REQDG_0004	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
MODBUS_Server_1_REQDG_0001	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
MODBUS_Server_1_REQDG_0002	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
MODBUS_Server_1_REQDG_0003	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	
byStatus	T_DIAG_MODBUS_ETH_MAPPING_STATUS	
eLastErrorCode	MASTER_ERROR_CODE	ERR_CONNECTION_TIMEOUT
eLastExceptionCode	MODBUS_EXCEPTION	NO_EXCEPTION
byDiag_3_reserved	BYTE	0
wCommCounter	WORD	116
wCommErrorCounter	WORD	49
MODBUS_Server_1_REQDG_0004	NXMODBUS_DIAGNOSTIC_STRUCTS.T_DIAG_MODBUS...	

Figure 42: ReqDiagnostics GVL in Online Mode

4.14.10. Prepare_Start Function

In this POU, the *PrepareStart* system event function is defined. It belongs to the communication task and is called before starting the application. When there is active communication with the PLC, it is possible to observe the event status and the call count in the *System Events* tab in the *Task Configuration* object. Every time the user starts the application, the count is incremented.

4.14.11. Prepare_Stop Function

In this POU, the *PrepareStop* system event function is defined. It belongs to the communication task and is called before stopping the application. When there is active communication with the PLC, it is possible to observe the event status and the call count in the *System Events* tab in the *Task Configuration* object. Every time the user stops the application, the count is incremented.

4.14.12. Start_Done Function

In this POU, the *StartDone* system event function is defined. It belongs to the communication task and is called when the application is successfully started. When there is active communication with the PLC, it is possible to observe the event status and the call count in the *System Events* tab in the *Task Configuration* object. Every time the user successfully launches the application, the count is incremented.

4.14.13. Stop_Done Function

In this POU, the *StopDone* system event function is defined. It belongs to the communication task and is called when the application is successfully stopped. When there is active communication with the PLC, it is possible to observe the event status and the call count in the *System Events* tab in the *Task Configuration* object. Every time the user successfully stops the application, the count is incremented.

5. Configuration

The Nexto Series CPUs are configured and programmed through the MasterTool IEC XE software. The configuration made defines the behavior and utilization modes for peripherals use and the CPUs special features. The programming represents the Application developed by the user.

5.1. Device

5.1.1. User Management and Access Rights

It provides functions to define users accounts and to configure the access rights to the project and to the CPU. Using the software MasterTool IEC XE, it's possible to create and manage users and groups, setting, different access right levels to the project.

Simultaneously, the Nexto CPUs have an user permissions management system that blocks or allows certain actions for each user group in the CPU. For more information, consult the MasterTool IEC XE User Manual MT8500 – MU299609, in the User Management and Access Rights section.

5.1.2. PLC Settings

On this tab of the generic device editor, you make the basic settings for the configuration of the PLC, for example the handling of inputs and outputs and the bus cycle task.

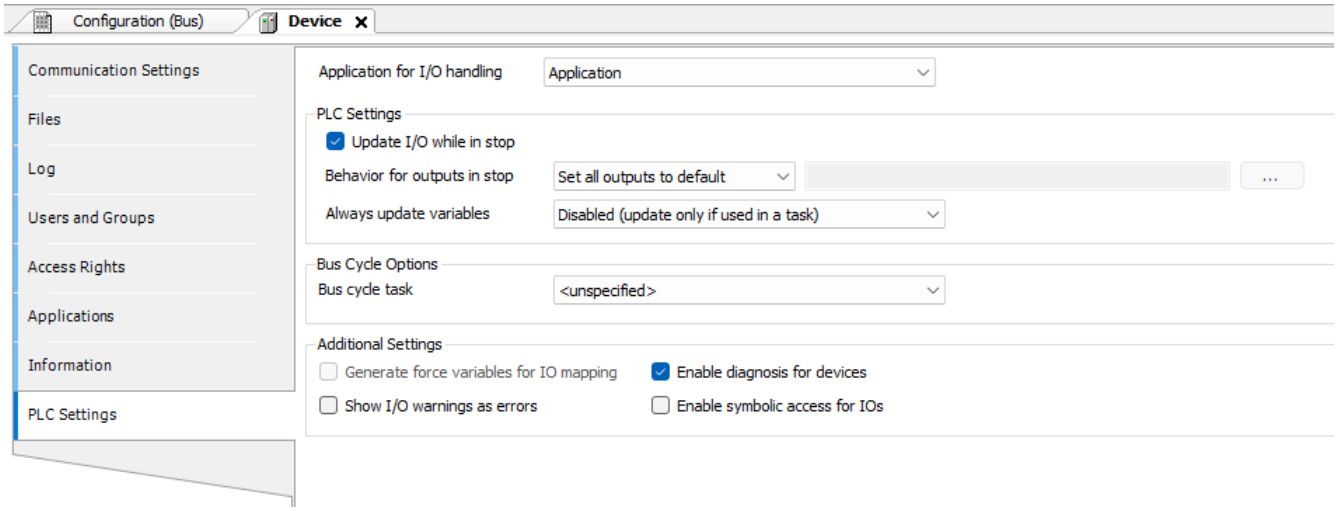


Figure 43: PLC Settings

Parameter	Description
Application for I/O handling	Application that is responsible for the I/O handling.
Refresh I/Os in stop	**TRUE:** The values of the input and output channels are also refreshed when the PLC is in STOP mode. If the watchdog detects a malfunction, the outputs are set to the predefined default values. **FALSE:** The values of the input and output channels in STOP mode are not refreshed.

Parameter	Description
Behavior of the outputs at stop	Handling of the output channels when the controller enters STOP mode: **Retain values:** The current values are retained. **All outputs to default value:** The default values resulting from the I/O mapping are assigned. **Execute program:** The handling of the output values is controlled by a program contained in the project which is executed in STOP mode. Enter the name of the program in the field on the right.
Always update variables	Globally defines whether or not the I/O variables are updated in the bus cycle task. This setting is effective for the I/O variables of the slaves and modules only if "deactivated" is defined in their update settings. **Deactivated (update only if used in a task):** The I/O variables are updated only if they are used in a task. **Enabled 1 (use bus cycle task if not used in any task):** The I/O variables in the bus cycle task are updated if they are not used in any other task. **Enabled 2 (always in bus cycle task):** All variables in each cycle of the bus cycle task are updated, regardless of whether they are used and whether they are mapped to an input or output channel.
Bus cycle task	Task that controls the bus cycle. By default the task defined by the device description is entered. By default, the bus cycle setting of the superordinate bus device applies (use cycle settings of the superordinate bus). This means that the device tree is searched upwards for the next valid definition of the bus cycle task.
Force variables for the I/O mapping	**TRUE:** When compiling the application, two global variables are created for each I/O channel which is mapped to a variable in the I/O Mapping dialog.
Activate diagnostics for devices	**TRUE:** The CAA Device Diagnosis library is integrated in the project. An implicit function block is generated for each device. If there is already a function block for the device, then either an extended function block is generated (example: EtherCAT) or another function block instance is added. This then contains a general implementation of the device diagnostics.
Display I/O warnings as errors	Warnings concerning the I/O configuration are displayed as errors.
Enable symbolic access for I/Os	**TRUE:** It allows access to I/O points from the internal symbolic name generated in the device declaration. The symbolic name can be consulted in the *Channel* column on the *Bus I/O Mapping* tab of each device.

Table 43: PLC Settings

ATTENTION

The Nexto (NX), Nexto Jet (NJ) and Xtorm (HX) products do not support the *Enable symbolic access for I/O* parameter.

5.2. CPU Configuration

5.2.1. General Parameters

The parameters related below are part of the CPU configuration included in the application. Each item must be properly verified for the correct project execution.

Besides these parameters, it is possible to change the name of each module inserted in the application by clicking the right button on the module. In the *Properties* item from the *Common* sheet, change the name, what is limited to 24 characters.

Settings	Description	Standard	Options
%Q Start Address	**Diagnostics Area (%Q)**		
	Starting address of the UCP diagnostics (%Q)	Automatically allocated on project creation.	0 to 97611
Size	Size of diagnostics area in bytes	693	It is not possible to change the size of the CPU diagnostics area
%Q Start Address	**Retaining Area (%Q)**		
	Starting address of the retentive data memory (%Q)	4096	0 to 98303
Size	Retain data memory size in bytes	98304	0 to 98304
%Q Start Address	**Persistent Area (%Q)**		
	Persistent data memory start address (%Q)	20480	0 to 98302
Size	Persistent data memory size in bytes	98304	0 to 98304
Start User Application after Reset by Watchdog	**CPU Parameters**		
	When enabled, starts the user application after resetting the hardware watchdog or restarting Runtime, but maintaining the diagnostic indication via WD LED and via variables.	Disabled	Enabled Disabled
Hot Swap Mode	Module hot swap mode	Enabled, no match consistency. (may vary according to CPU model)	- Denabled, only for declared modules - Disabled (with match consistency) - Disabled, no match consistency - Enabled, with match consistency only for declared modules - Enabled, with match consistency - Enabled, no match consistency

Settings	Description	Standard	Options
Enable I/O update per task	**Project Parameters**		
	Setting to update inputs and outputs in the tasks in which they are used.	unmarked	- Checked: Inputs and outputs are updated by the tasks in which they are used. - Unchecked: Inputs and outputs are only updated by MainTask
Enable retain and persistent variables in Function Blocks	Setting that allows the use of retentive and persistent variables in Function Blocks	unmarked	- Marked: allows the use of retentive and persistent variables in Function Blocks. - Unchecked: Exception error may occur at startup.

Table 44: CPU settings

Notes:

Generate error on tasks watchdog consistency: This parameter was discontinued as of MasterTool IEC XE version 1.32.

Enable I/O update per task: This parameter was added as of MasterTool IEC XE version 2.01.

ATTENTION

When the initial address or the retentive or persistent data memory size are changed in the user application, the memory is totally reallocated, what makes the retentive and persistent variable area be clean. So the user has to be careful so as not to lose the saved data in the memory.

ATTENTION

In situations where the symbolic persistent memory area is modified, a message will be displayed by MasterTool IEC XE programmer, to choose the behavior for this area after charging the modified program. The choice of this behavior does not affect the persistent area of direct representation, which is always clean.

ATTENTION

The option *Enable I/O update per task* is not supported for fieldbus masters such as NX5001 module. This feature is applicable only for input and output modules present on the controller local bus (main rack and expansion racks).

ATTENTION

Even when an I/O point is used in other tasks, with the *Enable I/O update per task* marked, it will continue to be updated in the MainTask as well; except when all the points of the module are used in some other task, in this case they will not be updated on MainTask anymore.

5.2.1.1. Hot Swap

Nexto Series CPUs have the possibility of I/O modules change in the bus with no need for system turn off and without information loss. This feature is known as hot swap.

CAUTION

Nexto Series CPUs do not guarantee the persistent and retentive variables retentivity in case the power supply or even the CPU is removed from the energized backplane rack.

On the hot swap, the related system behavior modifies itself following the configuration table defined by the user which represents the options below:

- Disable, for declared modules only
- Disabled (with startup consistency)
- Disabled, without startup consistency
- Enabled, with startup consistency for declared modules only
- Enabled, with startup consistency
- Enabled, without startup consistency

Therefore, the user can choose the behavior that the system must assume in abnormal bus situations and when the CPU is in *Run* Mode. The table below presents the possible abnormal bus situations.

Situation	Possible causes
Incompatible configuration	- Some module connected to the bus is different from the model that is declared in configuration.
Absent module	- The module was removed from the bus. - Some malfunctioning module is not responding to CPU - Some bus position is malfunctioning.

Table 45: Bus Abnormal Situations

For further information regarding the diagnostics correspondent to the above described situations, see *Diagnostics via Variables*.

If a module is present in a specific position in which should not exist according to the configuration modules, this module is considered as non-declared. The options of hot swap *Disabled, for Declared Modules Only* and *Enabled, with Startup Consistency for Declared Modules Only* do not take into consideration the modules that are in this condition.

5.2.1.1.1. Hot Swap Disabled, for Declared Modules Only

In this configuration, the CPU is immediately in *Stop* Mode when an abnormal bus situation (as described on Table 45) happens. The LED DG starts to blink 4x (according to Table 46). In this case, in order to make the CPU to return to the normal state *Run*, in addition to undo what caused the abnormal situation, it is necessary to execute a *Reset Warm* or a *Reset Cold*. If a *Reset Origin* is carried out, it will be necessary to perform the download so that the CPU can return to the normal state (*Run*). The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by MasterTool IEC XE in the *Online* menu.

The CPU will remain in normal *Run* even if find a module not declared on the bus.

5.2.1.1.2. Hot Swap Disabled

This setting does not allow any abnormal situation in the bus (as shown in Table 45) modules including undeclared and present on the bus. The CPU enters in *Stop* mode, and the DG LED begins to blink 4x (as in Table 46). For these cases, to turn the CPU back to normal *Run*, in addition to undo what caused the abnormal situation it is necessary to perform a *Reset Warm* or *Reset Cold*. If a *Reset Origin* is done, you need to download the project so that the CPU can return to normal *Run*. The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by MasterTool IEC XE in the *Online* menu.

5.2.1.1.3. Hot Swap Disabled, without Startup Consistency

Allows the system to start up even when some module is in an abnormal bus situation (as shown in Table 45). Abnormal situations are reported via diagnosis.

Any modification to the bus will cause the CPU to enter *Stop* Mode, and the DG LED will start blinking 2x (as in Table 46). In order for the CPU to return to the normal *Run* state in these cases, it is necessary to perform a *Reset Warm* or *Reset Cold*. If a *Reset Origin* is performed, it will be necessary to download the CPU so that the CPU can return to the normal *Run* state. The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by MasterTool IEC XE in the *Online* menu.

5.2.1.1.4. Hot Swap Enabled, with Startup Consistency for Declared Modules Only

"Startup" is the interval between the CPU energization (or reset command or application download) until the first time the CPU gets in *Run* Mode after been switched on. This configuration verifies if any abnormal bus situation has occurred (as described on Table 45) during the start. In affirmative case, the CPU gets in *Stop* Mode and the LED DG starts to blink 4x (according to Table 46). Afterwards, in order to set the CPU in *Run* mode, further to fix what caused the abnormal situation, it is necessary to execute a *Reset Warm* or *Reset Cold* command, which can be done by the MasterTool IEC XE (Online menu). If a *Reset Origin* is carried out, it will be necessary to perform the download so that the CPU can return to the normal state (*Run*).

After the start, if any module present any situation described in the previous table, the system will continue to work normally and will signalize the problem via diagnostics.

If there is no other abnormality for the declared modules, the CPU will go to the normal state (*Run*) even if a non-declared module is present on the bus.

ATTENTION

In this configuration when a power fault occurs (even temporally), *Reset Warm* Command,*Reset Cold* Command or a new application *Download* has been executed, and if any module is in an abnormal bus situation, the CPU will get into *Stop* Mode and the LED DG will start to blink 4x (according to Table 46). This is considered a startup situation.
This is the most advised option because guarantee the system integrity on its initialization and allows the modules change with a working system.

5.2.1.1.5. Hot Swap Enabled with Startup Consistency

This setting checks whether there has been any abnormal situation in the bus (as shown in Table 45) during the startup, even if there is no declared modules and present on the bus; if so, the CPU goes into *Stop* mode and the LED DG starts to blink 4x (as shown in Table 46). For these cases, to turn the CPU back to normal *Run*, in addition to undo what caused the abnormal situation it is necessary to perform a *Reset Warm* or *Reset Cold*. If a *Reset Origin* is done, you need to download the project so that the CPU can return to normal *Run*. The *Reset Warm*, *Reset Cold* and *Reset Origin* commands can be done by MasterTool IEC XE in the *Online* menu.

5.2.1.1.6. Hot Swap Enabled without Startup Consistency

Allows the system to start working even if a module is in an abnormal bus situation (as described on Table 45). The abnormal situations are reported via diagnostics during and after the startup.

ATTENTION

This option is advised for the system implementation phase as it allows the loading of new applications and the power off without the presence of all configured modules.

5.2.1.1.7. How to do the Hot Swap

CAUTION

Before performing the Hot Swap it is important to discharge any possible static energy accumulated in the body. To do that, touch (with bare hands) on any metallic grounded surface before handling the modules. Such procedure guaranties that the module static energy limits are not exceeded.

ATTENTION

It is recommended the hot swapping diagnostics monitoring in the application control developed by the user in order to guarantee the value returned by the module is validated before being used.

The hot swap proceeding is described below:

- Unlock the module from the backplane rack, using the safety lock.

- Take off the module, pulling firmly.
- Insert the new module in the backplane rack.
- Certify the safety lock is completely connected. If necessary, push the module harder towards to the backplane rack.

In case of output modules is convenient the points to be disconnected when in the changing process, in order to reduce the generation of arcs in module connector. This must be done by switching off the power supply or by forcing the output points using the software tools. If the load is small, there is no need for disconnecting.

It is important to note that in the cases the CPU gets in *Stop* Mode and the DG LED starts to blink 4x (according to Table 46, due to any abnormal bus situation (as described on Table 45, the output modules have its points operation according to the module configuration when CPU toggles from *Run* Mode to *Stop* Mode. In case of application startup, when the CPU enters *Stop* Mode without having passed to the *Run* Mode, the output modules put their points in failure secure mode, in other words, turn it off (0 Vdc).

Regarding the input modules, if one module is removed from energized backplane rack, the logic point's state will remain in the last value. In the case a connector is removed, the logic point's state will be put in a safe state, it means zero or high impedance.

ATTENTION

Always proceed to the substitution of one module at a time for the CPU to update the modules state.

Below, Table 46 presents the bus conditions and the Nexto CPU DG LED operation state. For further information regarding the diagnostics LEDs states, see Diagnostics via LED section.

Condition	Enabled, with Startup Consistency	Enabled, with Startup Consistency for Declared Modules Only	Enabled, without Startup Consistency	Disabled	Disabled, for declared modules only	Disabled, without Startup Consistency
Non declared module	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Stop
Non declared module (startup condition)	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run
Absent module	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Stop
Absent module (startup condition)	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run
Incompatible configuration	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 2x Application: Run	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Stop

Condition	Enabled, with Startup Consistency	Enabled, with Startup Consistency for Declared Modules Only	Enabled, without Startup Consistency	Disabled	Disabled, for declared modules only	Disabled, without Startup Consistency
Incompatible configuration(startup condition)	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run or LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Run
Duplicated slot address	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Stop
Non-operational module	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 4x Application: Stop	LED DG: Blinks 2x Application: Stop

Table 46: Hot Swap and Conditions Relations

Note:

Enabled, without startup consistency: When this hot-swap mode is configured, in normal situations when there's an incompatible module on the system's startup, the application will go from Stop to Run. However, if that module is configured as a NX5000 or a NX5001 and there's a different module in that position, the application will stay in Stop.

5.2.1.2. Retain and Persistent Memory Areas

The Nexto CPU allows the use of symbolic variables and output variables of direct representation as retentive or persistent variables.

The output variables of direct representation which will be retentive or persistent must be declared in the CPU *General Parameters*, as described at CPU Configuration. Symbolic names also can be attributed to these output variables of direct representation using the AT directive, plus using the key word RETAIN or PERSISTENT on its declaration. For example, being %QB4096 and %QB20480 within the retentive and persistent memory, respectively:

```
PROGRAM UserPrg
VAR RETAIN
byRetentiveVariable_01 AT %QB4096 : BYTE;
END_VAR
VAR PERSISTENT
byPersistentVariable_01 AT %QB20480 : BYTE;
END_VAR
```

In case the symbolic variables declared with the AT directive are not inside the respective retentive and/or persistent memory, errors during the code generation in MasterTool can be presented, informing that there are non-retentive or non-persistent variables defined in the retentive or persistent memory spaces.

Regarding the symbolic variables which will be retentive or persistent, only the retentive variables may be local or global, as the persistent symbolic variables shall always be global. For the declaration of retentive symbolic variables, it must be used the key word *RETAIN*. For example, for local variables:

```
PROGRAM UserPrg
VAR RETAIN
  wLocalSymbolicRetentiveVariable_01 : WORD;
END_VAR
```

Or, for global variables, declared within a list of global variables:

```
VAR_GLOBAL RETAIN
  wGlobalSymbolicRetentiveVariable_01 : WORD;
END_VAR
```

On the other hand, the persistent symbolic variables shall be declared in a Persistent Variables object, being added to the application. These variables will be global and will be declared in the following way within the object:

```
VAR_GLOBAL PERSISTENT RETAIN
  wGlobalSymbolicPersistentVariable_01 : WORD;
END_VAR
```

As of versions 1.5.1.1 the Nexto Series CPUs allow flexibility on the usage of retentive and persistent memories. This means that the user will be able to choose the size that will be used for each type of memory, as long as the retentive and persistent memory sum don't exceed the total limit available in each CPU model. The total of retentive and persistent memory available is described in the Table 7 in Memory.

If the retentive symbolic, persistent symbolic, retentive %Q and persistent %Q memory sum exceed the total available, MasterTool will show an error during the code generation.

```
VAR_GLOBAL PERSISTENT RETAIN
  wGlobalSymbolicPersistentVariable_01 : WORD;
END_VAR

VAR_GLOBAL RETAIN
  wGlobalSymbolicRetentiveVariable_01 : WORD;
END_VAR
```

ATTENTION

To use the retentive and persistent memory flexibly, it's necessary to use MasterTool IEC XE 2.03 or higher.

5.2.1.3. Project Parameters

The CPU project parameters are related to the configuration for input/output refreshing at the task that they are used of the project tasks and the options for reading and writing on the memory card.

Configuration	Description	Default	Options
Enable I/O update per task	Updates the input and output in the tasks where they are used	Unmarked	- Marked - Unmarked
Enable retain and persistent variables in Function Blocks	Setting to allow the use of retentive and persistent variables in function blocks	Unmarked	- Marked - Unmarked
Copy Project from CPU to Memory Card	**Memory Card**		
	Copy the project from the CPU internal memory to the memory card	Disabled	- Enabled: Configuration enabled - Disabled: Configuration disabled
Password to Copy Project from CPU to Memory Card	Password for coping the project from the CPU internal memory to memory card	-	6 digits password (0 to 999999)
Copy Project from Memory Card to CPU	Copy the project from the memory card to the CPU internal memory	Disabled	- Enabled: Configuration enabled - Disabled: Configuration disabled
Password to Copy Project from Memory Card to CPU	Password for coping the project from the memory card to the CPU internal memory	-	6 digits password (0 to 999999)

Table 47: CPU Project Parameters

ATTENTION

After setting the project copy possibilities and having created the boot application, it must be found the "*Application.crc*" file in order the configurations concerning the memory card have effect. The search can be done at *Select the Application.crc* through the *Find File...* key, as can be seen on Figure 131.

5.2.2. External Event Configuration

The external event is a feature available in the CPU which enables a digital input, configured by the user, when activated, triggers the execution of a specific task with user-defined code. Thus, it is possible that through this input, when triggered, interrupt the execution of the main application and run the set application in the task *ExternInterruptTask00*, which has higher priority than other application tasks. Because the inputs and outputs are updated in the context of the MainTask task, the External Event task does not have the input and output data updated at the time of its call. If necessary, use the I/O update functions.

It is also important to note that, to avoid the generation of several events in a very short space of time, that was limited the treatment of this type of event in every 10 ms, i.e., if two or more events occurs during 10 ms after the first event, the second and subsequent events are discarded. This limitation is imposed to prevent an external event that is generated in an uncontrolled way, do not block the CPU, since the task has a higher priority over the others.

To configure an external event is necessary to insert a digital input module and perform the configurations described below, in the CPU, through the MT8500 programming tool software.

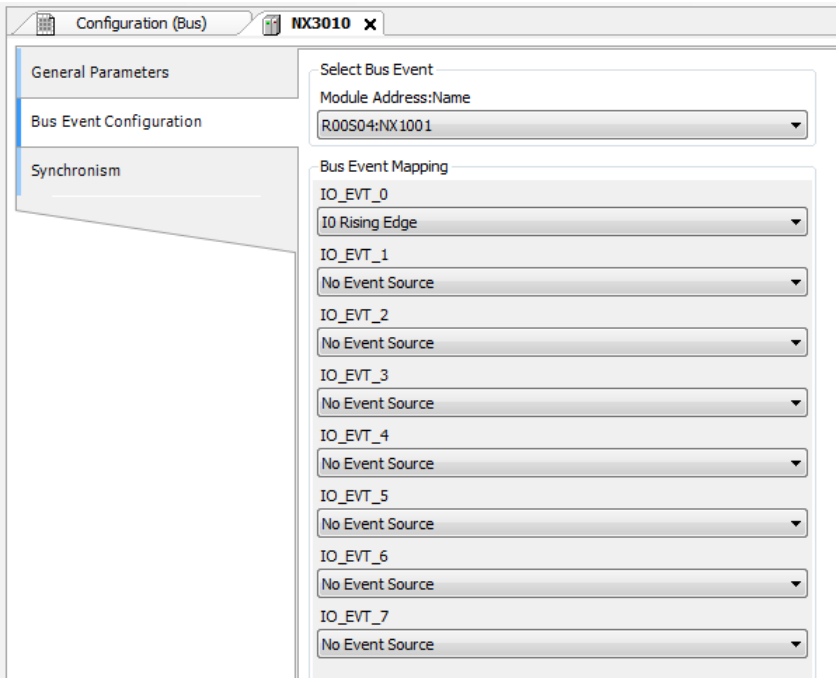


Figure 44: Configuration Screen for External Event in CPU

In the configuration external event tab, within the CPU settings, it is necessary to select which module will be the interruption source, in the field *Module Address: Name*. Then it must be selected which input of this module will be responsible for the event generation (*IO_EVT_0*). In this selection the options described in the figure below can be chosen.

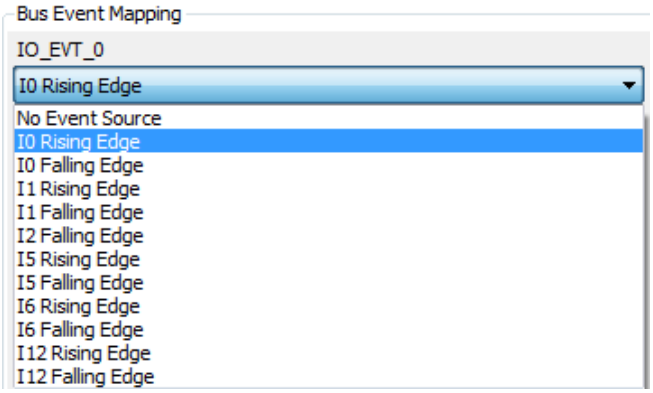


Figure 45: NX1001 Module External Event Source Options

In addition to configuring the CPU it is required to configure the task responsible for executing user-defined actions. In this case the user must use a project profile that supports external events. For further information see the section Project Profiles. In the configuration screen of the *ExternInterruptTask00* (figure below), it is necessary to select the event source in the corresponding field. In this case, *IO_EVT_0* should be selected since the other origin sources (*IO_EVT_1* to *IO_EVT_7*) are not available. In the sequence, the field *POU* should be checked if the right POU is selected, because it will be used by the user to define the actions to be performed when an external event occurs.

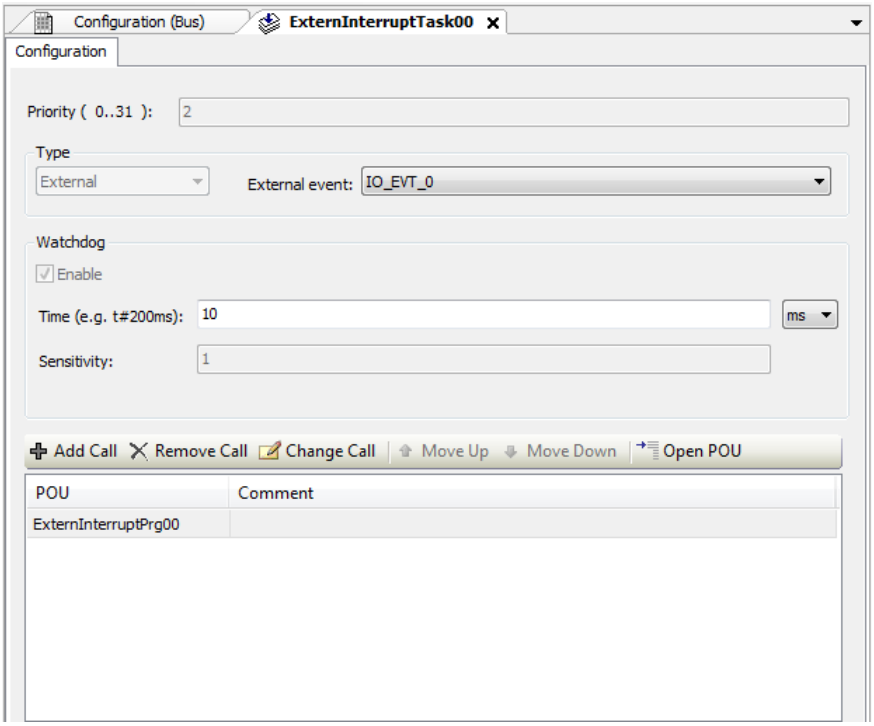


Figure 46: ExternInterruptTask00 Configuration Screen

5.2.3. SOE Configuration

The SOE (*Sequence of Events*) is responsible for the generation of a sequence of digital events. Through the SOE it is possible to analyze the historic behavior of the system variables mapped in its monitoring area. The SOE is an exclusive service available for the NX3020 and NX3030 models.

Once the SOE service has been enabled, the CPU starts to behave as a DNP3 server, thus it is necessary the support to the DNP3 protocol by the client for the use of this resource. The supported object types as well as the function codes and the qualifiers can be found at Annex. DNP3 Interoperability.

The SOE service uses the %Q addresses in order to form its base of static data. For it, it has to be set a continuous area of %Q memory where the user will inform its beginning and size. For redundant projects the %Q area also has to be redundant so that in the switchover moment the DNP3 server data base is kept.

The DNP3 first object address will always be 0, corresponding to %QBxxxx's bit 0, where xxxx is the %Q initial address.

Thus, once defined the static data base, the user must copy each digital point which should generate events within the %Q continuous area. The maximum number of points which can be copied is 8000.

For the events configuration, it is necessary to inform only the size of the events queue. The SOE uses a special and dedicated queue (not the one described on Protocols Configuration section), which is persistent and redundant, so the events will not be lost in the switchover moment neither in case of a power supply failure. In case an overflow occurs in the events queue, the oldest events will be overwritten. In case in one single cycle are generated more events than what is supported by the queue, its generation is interrupted and the overflow diagnostic is turned on (SOE[x].bOverflowStatus). For example, if 100+n bits vary in a 100 events configuration, causing a dispose of n events.

The SOE will run in the MainTask context, starting already at the first cycle. The SOE will run at the end of each MainTask cycle, comparing the mapped bits in order to detect transitions occurred in the cycle. In this way, every cycle in which the events are generated, an increase of time in this cycle of the MainTask will occur. In the worst case (1000 events, being generated only 1000 and discarded the remaining ones), this influence will be approximately of 5 ms. Therefore, for an application with the SOE enabled, the user will have to take into account this time when setting the parameters of watchdog time and interval of the MainTask.

For the use of it the user must set the following parameters in the SOE Configuration tab:

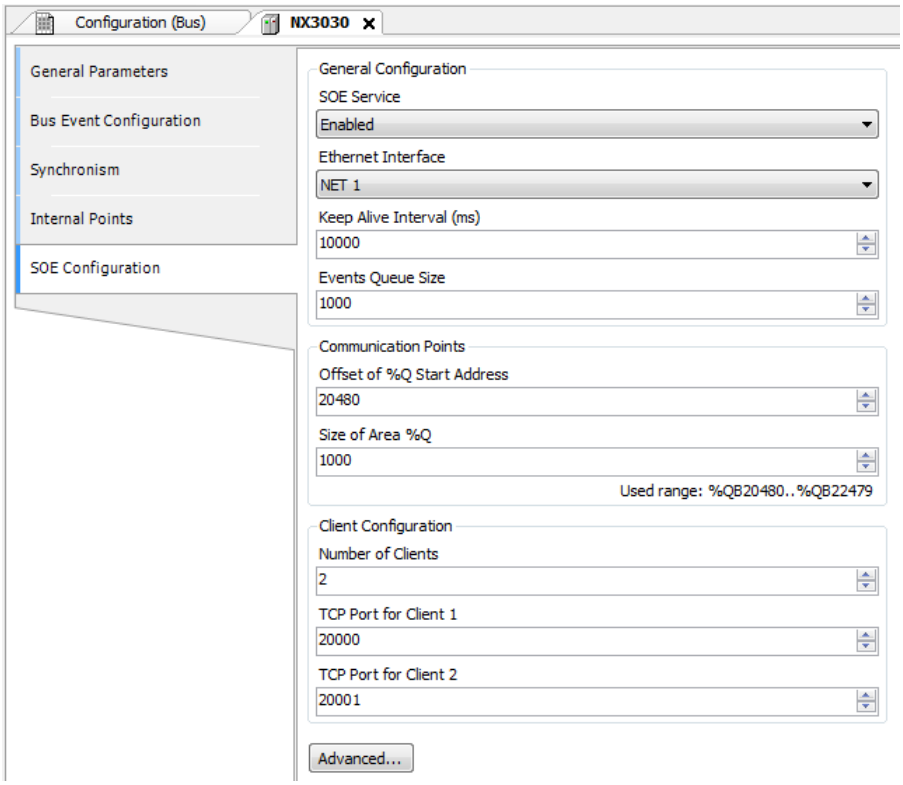


Figure 47: Events Sequence Configuration

Configuration	Description	Default Value	Options
SOE Service	**General Configuration**		
	Enables the SOE	Disabled	Enabled Disabled
Ethernet Interface	Selects the used interface	NET 1	NET 1 NET 2
Keep Alive Interval (ms)	Keep alive (ms) interval messages	10000	0 to 4294967295
Events Queue Size	Events queue size	1000	100 to 1000
Offset of %Q Start Address	**Communication Points**		
	Initial address for static data	20480	Any %Q area address can be used
Size of Area %Q	Memory size to be used by the static data (%Q)	1000	1 to 1000
Number of Clients	**Client Configuration**		
	Defines the number of clients	2	1, 2
TCP Port for Client 1	Selects the communication port for the first client	20000	1 to 65535
TCP Port for Client 2	Selects the communication port for the second client	20001	1 to 65535

Table 48: SOE Configuration

Notes:

Data Memory Size: The data memory size reserved to be used by the static data will always be twice the value set as the second half of the memory area is used to store the previous variables values of the first half.

Keep Alive: While it is connected to a client, keep alive messages will be sent in intervals according to what has been set. If the client does not respond to these messages, the connection is closed. That is, a connection between client and server may take a time equal to the interval set to be closed in case of error.

In the advanced options (*Advanced...* key) it is possible to set the communication addresses regarding to the DNP3 protocol.

Configuration	Description	Default Value	Options
DNP3 Source Address	Origin Address (PLC)	4	0 to 65519
DNP3 Destination Address of Client 1	Address of the first client	3	0 to 65519
DNP3 Destination Address of Client 2	Address of the second client	3	0 to 65519

Table 49: SOE Advanced Configurations

Note:

DNP3 Address: The DNP3 addresses from the range 65520 to 65535 cannot be set at the origin or at a destiny as they are used for messages in broadcast.

ATTENTION

The DNP3 *Data Link* messages are not used by the Nexto Series CPUs as the standard does not recommend its use them in TCP/IP communications.

5.2.4. Time Synchronization

For the time synchronization, Nexto Series CPUs use the SNTP (*Simple Network Time Protocol*) or the synchronism through IEC 60870-5-104.

To use the time sync protocols, the user must set the following parameters at *Synchronism* tab, accessed through the CPU, in the device tree:

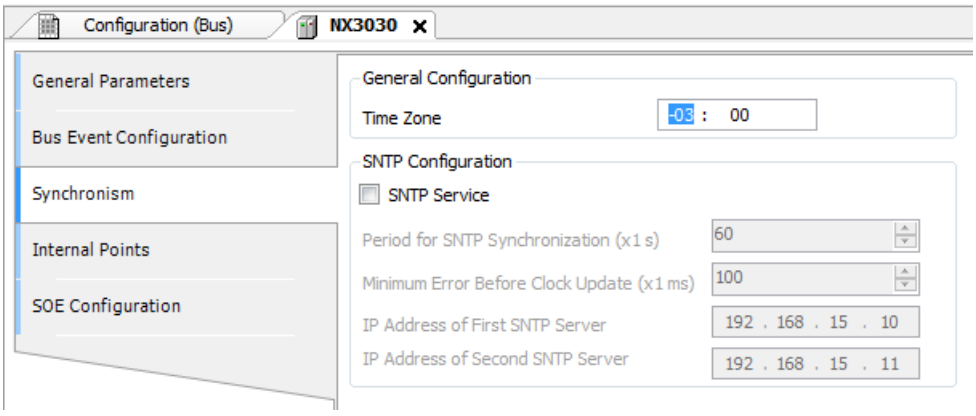


Figure 48: SNTP Configuration

Configuration	Description	Default	Options
Time Zone (hh:mm)	Time zone of the user location. Hours and minutes can be inserted.	-3:00	12:59 to +13:59
SNTP Service	Enables the SNTP service.	Disabled	Disabled Enabled
Period for SNTP Synchronization (x1 s)	Time interval of the synchronization requests (seconds).	60	1 to 255
Minimum Error Before Clock Update (x1 ms)	Offset value acceptable between the server and client (milliseconds).	100	1 to 65519
IP Address of First SNTP Server	IP Address of the primary SNTP server.	192.168.15.10	1.0.0.1 to 223.255.255.254
IP Address of Second Second SNTP Server	IP Address of the secondary SNTP server.	192.168.15.11	1.0.0.1 to 223.255.255.254

Table 50: SNTP Configurations

Notes:

SNTP Server: It is possible to define a preferential address and another secondary one in order to access a SNTP server and, therefore, to obtain a synchronism of time. If both fields are empty, the SNTP service will remain disabled.

Time zone: The time zone configuration is used to convert the local time into UTC and vice versa. While some sync sources use the local time (IEC 60870-5-104 protocol, SetDateAndTime Function), others use the UTC time (SNTP). The UTC time is usually used to stamp events (DNP3, IEC 60870-5-104 and MasterTool Device Log), while the local time is used by an others CPU's features (GetDateAndTime function, OTD date and time info).

It is allowed to enable more than one sync source on the project, however the device doesn't supports the synchronism from more than one sync source during operation. Therefore there are implicitly defined a priority mechanism. The synchronism through SNTP is more prioritary than through IEC 60870-5-104 protocol. So, when both sources are enabled and SNTP server is present, it is going to be responsible for the CPU's clock sync, and any sync command from IEC 60870-5-104 is going to be denied.

5.2.4.1. IEC 60870-5-104

In case the synchronism is through IEC 60870-5-104 protocol, the user must enable the time sync at the protocol configuration screen to receive the clock synchronization. To set this option on the device, check the parameter *Enable Time Synchronization* available at the Application Layer section.

ATTENTION

If the PLC receives a time sync command from the control center, and this option is disabled, an error answer will be returned to that command. But if this option is enabled then a success message will be returned to the control center, even that the sync command be discarded for there is another synchronism method active with higher priority.

This synchronism method should be used only as an auxiliary synchronism method, once the precision of the clock sync process depends a lot on delays and traffic on the network, as well as the processor load on the CPU, as this mechanism is treated by a low priority task.

ATTENTION

In redundant PLCs architectures, the IEC 60870-5-104 Server driver is disabled on non-active PLC. This way, isn't recommended the use of this synchronism method in redundant systems. Because the non-active PLC might take several seconds after the switchover until its clock is synchronized. To redundant systems it is recommended the use of SNTP.

5.2.4.2. SNTP

When enabled, the CPU will behave as a SNTP client, which is, it will send requests of time synchronization to a SNTP/NTP server which can be in the local net or in the internet. SNTP client works with a resolution of 1 ms, but with an accuracy of 100 ms. The precision of the time sync through SNTP depends on the protocol configurations (minimum error to clock update) and the features of the Ethernet network where it is, if both client and server are in the same network (local) or in different networks (remote). Typically the precision is in tens of milliseconds order.

The CPU sends the cyclic synchronization requests according to the time set in the *Period for SNTP Synchronization* field. In the first synchronization attempt, just after the service start up, the request is for the first server set in the first server IP address. In case it does not respond, the requests are directed to the second server set in the second server IP address providing a redundancy of SNTP servers. In case the second server does not respond either, the same process of synchronization attempt is performed again but only after the Period of Synchronization having been passed. In other words, at every synchronization period the CPU tries to connect once in each server, it tries the second server in case the first one does not respond. The waiting time for a response from the SNTP server is defined by default in 5 s and it cannot be modified.

If, after a synchronization, the difference between the current time of the CPU and the one received by the server is higher than the value set in the *Minimum Error Before Clock Update* parameter, the CPU time is updated. SNTP uses the time in the UTC (Universal Time Coordinated) format, so the *Time Zone* parameter needs to be set correctly so the time read by the SNTP will be properly converted to a local time.

The execution process of the SNTP client can be exemplified with the following steps:

1. Attempt of synchronization through the first server. In case the synchronization occurs successfully, the CPU waits the time for a new synchronization (*Period for SNTP Synchronization*) and will synchronize again with this server, using it as a primary server. In case of failure (the server does not respond in less than 5 s) step 2 is performed.
2. Attempt of synchronization through the second server. In case the synchronization occurs successfully, the CPU waits the time for a new synchronization (*Period for SNTP Synchronization*) and will try to synchronize with this server using the primary server. In case of failure (the server does not respond in less than 5 s) the time relative to the Synchronization Period is waited and step 1 is performed again.

As the waiting time for the response of the SNTP server is 5 s, the user must pay attention to lower than 10 s values for the Synchronization Period. In case the primary server does not respond, the time for the synchronization will be the minimum of 5 s (waiting for the primary server response and the synchronization attempt with secondary server). In case neither the primary server nor the secondary one responds, the synchronization time will be 10 s minimum (waiting for the two servers response and the new connection with first server attempt).

Depending on the SNTP server's subnet, the client will use the Ethernet interface that is in the corresponding subnet to make the synchronization requests. If there is no interface configured on the same subnet as the server, the request can be made by any interface that can find a route to the server.

ATTENTION

The SNTP Service depends on the user application only for its configuration. Therefore, this service will be executed even when the CPU is in *STOP* or *BREAKPOINT* modes, as long as there is an application in the CPU with the SNTP client enabled and correctly configured.

CAUTION

It is vital to setup at least one SNTP server. It is recommended to configure two SNTP servers (primary and secondary). SNTP synchronism is necessary to generate events with a coherent time stamp between the CPA and CPB and with the world time. Another usefulness is to avoid discontinuities during a switchover in applications that reference date and time, considering that there is no synchronization of date and time between the PLCs through the NETA and NETB synchronism channels.

5.2.4.3. Daylight Saving Time (DST)

The DST configuration must be done indirectly through the function *SetTimeZone*, which changes the time zone applied to the RTC. In the beginning of the DST, it has to be used a function to increase the time zone in one hour. At the end of the DST, it is used to decrease it in one hour.

For further information, see the section RTC Clock.

5.2.5. Internal Points

A communication point is storage on the CPU memory under form of two distinct variables. One represents the point's value (type BOOL, BYTE, WORD, etc...), while another, represents its quality (type QUALITY). Internal Points are those which the value and the quality are calculated internally by the user application, that is, they don't have an external origin like occur with points linked to IEDs (Communication drivers of type Master/Client) or to local I/O modules.

ATTENTION

Different from what happen with I/O modules declared on local bus, which have its own quality variables created by MasterTool (IOQualities GVL) and auto updated by the CPU, I/O modules declared on PROFIBUS remotes don't have.
It is user responsibility to declare PROFIBUS point's quality variables, the association of these quality variables with the value variables at Internal Points tab, as well as generation and update of the quality variables value, from the existents PROFIBUS diagnostics: PROFIBUS I/O modules, PROFIBUS head and PROFIBUS Master.

This *Internal Points* configuration tab's function is to relate the variable which represents a point's value with the one which represents its quality. It must be used to relate value and quality variables internally created on the PLC program (as in a GVL), which ones typically will be posteriorly mapped to a communication driver, of type Server, for communication with the control center.

ATTENTION

If a value variable doesn't own a related quality variable, it will be reported as default a constant good quality (no significant indication) when the value variable is reported to a client or control center.

In this way, this tab purpose isn't to create or declare internal points. To do that, just declare value and/or quality variables in a GVL and map it on the communication driver.

The internal points configuration, viewed on the figure below, follow the parameters described on table below. It's possible to configure up to 5120 entries on *Internal Points* table.

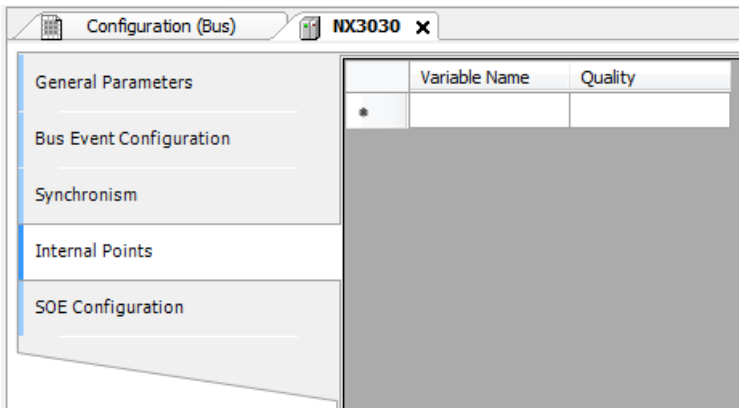


Figure 49: Internal Points Configuration Screen

Configuration	Description	Default	Options
Variable Name	Symbol variable which storage the internal point value	-	Accept variables of type BOOL, WORD, DWORD, LWORD, INT, DINT, LINT, UINT, UDINT, ULINT, REAL, LREAL or DBP. The variable can be simple, array or array's element and can be part of a struct.
Quality	Symbol variable which storage the internal point quality	-	QUALITY type variables (LibRtuStandard), which can be simple, array or array's element and can be part of a struct.

Table 51: Internal Points Configuration

The figure below show an example of two internal points configuration:

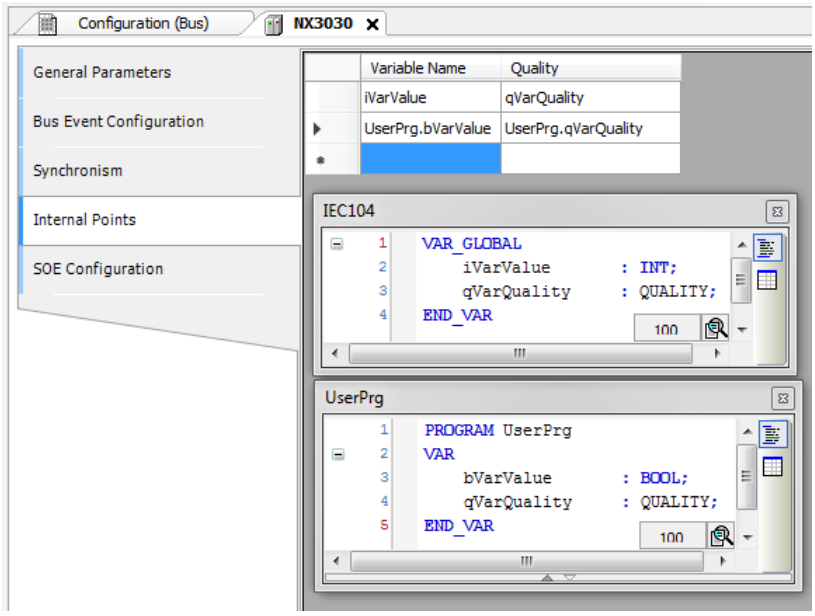


Figure 50: Internal Points Configuration Example

5.2.5.1. Quality Conversions

The internal point's quality is a trust level information about the value stored on that point. The quality may inform, for example, that the value stored is out of range, or yet that it is valid, but low trusted.

The Standards IEC 61850, DNP3 and IEC104 have their own formats to representation of point's quality information. The Nexto Series, by its turn, have its own quality format (but quite similar to IEC 61850) called *Internal Quality*. This format is defined by type QUALITY (library LibRtuStandard) and it is used internally to quality storage, allowing to be done conversion between protocols without information loss.

When it is done a mapping of a same communication point between two drivers, the quality conversion is automatically realized in two steps. For example: in case a communication point is mapped from a DNP3 Client driver to a IEC104 Server driver, first the quality will be converted from DNP3 format to internal format (and stored internally in the CPU), after that it will be converted from the internal format to IEC104 format.

The following tables define the protocols own formats conversion to internal format. Case it is necessary to consult the conversion between protocols, it is needed to analyze in two steps, looking each of the tables to internal format and after correlating them.

ATTENTION

In case of internal points mapped to communication drivers, it is not recommended to modify the value of quality flags that dont have a correspondent on the given protocol (i.e, flags that are not described on the following tables). This will result on generation of events equal to the previous one (but with a more recent timestamp) and, this way, depending on the configuration selected for the transmission mode of analog inputs events, it could overwrite the previous event if this one was not delivered to the control center yet.

5.2.5.1.1. Internal Quality

This is the QUALITY structure. The table shows detailed each of its components.

Bit	Name	Type	Description
0	FLAG_RESTART	BOOL	The RESTART flag indicates that the data haven't been updated by the field since the device's reset.
1	FLAG_COMM_FAIL	BOOL	Indicates there is a communication failure on the way between the data origin device and the reports device.
2	FLAG_REMOTE_SUBSTITUTED	BOOL	If TRUE the data values are overwritten in the remote communication devices.
3	FLAG_LOCAL_SUBSTITUTED	BOOL	If TRUE the data value is overwritten by the device which generated this flag. This behavior might occur due to a working in diagnostic or temporary due to human intervention.
4	FLAG_FILTER	BOOL	Flag used to signalize and prevent the event communication channel overload, as oscillations (rapid changes) on the digital inputs.
5	FLAG_OVERFLOW	BOOL	This flag should indicates a quality problem, that the value, of the attribute to which the quality has been associated, is beyond representation.
6	FLAG_REFERENCE_ERROR	BOOL	This flag should identify that the value cannot be correct due to out of calibration reference.
7	FLAG_INCONSISTENT	BOOL	This flag should identify that an evaluation function has found an inconsistency.
8	FLAG_OUT_OF_RANGE	BOOL	This flag should indicates a quality problem that the attribute to which the quality has been associated is beyond the predefined values capacity.
9	FLAG_INACCURATE	BOOL	This flag should indicates that the value doesn't attend the declared precision of the source.
10	FLAG_OLD_DATA	BOOL	A value seems to be outdated. In case an update doesn't occur during a specific time period.
11	FLAG_FAILURE	BOOL	This flag should indicates that a watch function detected an internal or external failure.
12	FLAG_OPERATOR_BLOCKED	BOOL	Update blocked by operator.

Bit	Name	Type	Description
13	FLAG_TEST	BOOL	This must be an additional identifier which can be used to classify a value being that a test value which won't be used to operational ends.
14-15	RESERVED	-	Reserved
16-17	VALIDITY	QUALITY_VALIDITY	0 – Good (Trustfull value, means that there is no abnormal conditions) 1 – Invalid (Value doesn't match the IED's value) 2 – Reserved (Reserved) 3 – Questionable (Present value might be not the same from the IED)

Table 52: QUALITY Structure

5.2.5.1.2. IEC 60870-5-104 Conversion

The tables below presents the digital, analog, Step Position, Bitstring and counters internal point's conversion to IEC 60870-5-104 of Nexto Series available to MT8500.

Internal Points -> IEC 60870-5-104 Digital		
Internal Quality		
Flags	VALIDITY	IEC 60870-5-104 Quality
FLAG_RESTART	ANY	NOT TOPICAL
FLAG_COMM_FAIL	ANY	NOT TOPICAL
FLAG_REMOTE_SUBSTITUTED	ANY	SUBSTITUTED
FLAG_LOCAL_SUBSTITUTED	ANY	SUBSTITUTED
FLAG_FILTER	ANY	-
FLAG_OVERFLOW	ANY	-
FLAG_REFERENCE_ERROR	ANY	-
FLAG_INCONSISTENT	ANY	-
FLAG_OUT_OF_RANGE	ANY	-
FLAG_INACCURATE	ANY	-
FLAG_OLD_DATA	ANY	NOT TOPICAL
FLAG_FAILURE	ANY	INVALID
FLAG_OPERATOR_BLOCKED	ANY	BLOCKED
FLAG_TEST	ANY	-
ANY	VALIDITY_INVALID	INVALID

Table 53: Digital Points Conversion Internal to IEC 60870-5-104

Internal Points -> IEC 60870-5-104 Analog, Step Position and Bitstring		
Internal Quality		
Flags	**VALIDITY**	**IEC 60870-5-104 Quality**
FLAG_RESTART	ANY	NOT TOPICAL
FLAG_COMM_FAIL	ANY	NOT TOPICAL
FLAG_REMOTE_SUBSTITUTED	ANY	SUBSTITUTED
FLAG_LOCAL_SUBSTITUTED	ANY	SUBSTITUTED
FLAG_FILTER	ANY	-
FLAG_OVERFLOW	ANY	OVERFLOW
FLAG_REFERENCE_ERROR	ANY	INVALID
FLAG_INCONSISTENT	ANY	INVALID
FLAG_OUT_OF_RANGE	ANY	OVERFLOW
FLAG_INACCURATE	ANY	INVALID
FLAG_OLD_DATA	ANY	NOT TOPICAL
FLAG_FAILURE	ANY	INVALID
FLAG_OPERATOR_BLOCKED	ANY	BLOCKED
FLAG_TEST	ANY	-
ANY	VALIDITY_INVALID	INVALID

Table 54: Analog, Step Position and Bitstring Points Conversion Internal to IEC 60870-5-104

Internal Points -> IEC 60870-5-104 Counters		
Internal Quality		
Flags	**VALIDITY**	**IEC 60870-5-104 Quality**
FLAG_RESTART	ANY	-
FLAG_COMM_FAIL	ANY	-
FLAG_REMOTE_SUBSTITUTED	ANY	-
FLAG_LOCAL_SUBSTITUTED	ANY	-
FLAG_FILTER	ANY	-
FLAG_OVERFLOW	ANY	OVERFLOW
FLAG_REFERENCE_ERROR	ANY	-
FLAG_INCONSISTENT	ANY	-
FLAG_OUT_OF_RANGE	ANY	-
FLAG_INACCURATE	ANY	-
FLAG_OLD_DATA	ANY	-
FLAG_FAILURE	ANY	INVALID
FLAG_OPERATOR_BLOCKED	ANY	-
FLAG_TEST	ANY	-
ANY	VALIDITY_INVALID	INVALID

Table 55: Counters Conversion Internal to IEC 60870-5-104

5.2.5.1.3. MODBUS Internal Quality

As the MODBUS standard don't specify quality types to each point, but for help on use of each point's communication diagnostic, MasterTool allows the quality variables mapping, through an internal own structure, to each MODBUS point. The table below describes the quality types that each MODBUS point can assume.

Resulting Quality	Resulting VALIDITY	Description
FLAG_RESTART	VALIDITY_INVALID	Initial value. The point was never updated.
-	VALIDITY_GOOD	Communication OK. The point is updated.
FLAG_COMM_FAIL AND FLAG_RESTART	VALIDITY_INVALID	Communication error. The point never was updated.
FLAG_COMM_FAIL AND FLAG_OLD_DATA	VALIDITY_QUESTIONABLE	An error has occurred but the point was updated and now has an old value.
FLAG_FAILURE AND FLAG_RESTART	VALIDITY_INVALID	It has received an exception response and the point kept its initial value.
FLAG_FAILURE AND FLAG_OLD_DATA	VALIDITY_QUESTIONABLE	It has received an exception response, but the point has a valid old value.
FLAG_RESTART AND FLAG_OLD_DATA	VALIDITY_QUESTIONABLE	Device stopped. The point has an old value.

Table 56: MODBUS Quality

5.2.5.1.4. Local Bus I/O Modules Quality

To help in the use of each I/O point's diagnostic, MasterTool automatically creates a quality structure to each local bus module used on the PLC project, through an own internal structure accessible by structure QUALITY, available in GVL *IOQualities*.

The table below describes the quality types to each input and output point.

For further information look at GVL IOQualities.

Diagnostics	Resulting Quality	Resulting VALIDITY	Description
Don't care	FLAG_RESTART	VALIDITY_INVALID	The quality has this value before have been read or written for the first time.
None	-	VALIDITY_GOOD	Communication OK. The point is updated.
None	FLAG_OLD_DATA AND FLAG_FAILURE	VALIDITY_QUESTIONABLE	Non-operational module. However, the data have been read or written at least once.
bOverRange OR bUnderRange	FLAG_OUT_OF_RANGE	VALIDITY_INVALID	The value is above or below the module's input allowed range.
bInputNotEnable OR bOutputNotEnable	FLAG_OPERATOR_BLOCKED	VALIDITY_INVALID	Input/Output not enable.
bOpenLoop	FLAG_FAILURE	VALIDITY_INVALID	Open loop in input module.
bFatalError	FLAG_FAILURE	VALIDITY_INVALID	Hardware fatal failure.
bNoExternalSupply	FLAG_FAILURE	VALIDITY_INVALID	External power supply is under operational minimum limit.
bShortCircuit OR bOutputShortCircuit	FLAG_FAILURE	VALIDITY_INVALID	Output short-circuit.
bCalibrationError	FLAG_INACCURATE	VALIDITY_INVALID	Calibration error.
bColdJunctionSensorError	FLAG_INACCURATE	VALIDITY_INVALID	Cold junction sensor error.

Table 57: I/O Modules Quality

5.2.5.1.5. PROFIBUS I/O Modules Quality

Different from local bus, MasterTool doesn't automatically create the PROFIBUS modules quality structures, and neither the PLC update such structures. Therefore the creation and cyclic update of PROFIBUS modules quality is user responsibility.

To help on the development of such applications, there are following practical examples, in ST language, for the main PROFIBUS modules (DI, DO, AI, AO), based on Nexto Serie's PROFIBUS slaves (NX5110). The user should feel encouraged to make any needed adaptation and change to fit to its application.

ATTENTION

For the routines, presented in sequence, correct functioning it is necessary to enable *Status in Diagnose* in the PROFIBUS slaves.

The development of PROFIBUS I/O modules quality points update routine must began from quality variables declaration and initialization, from a GVL:

```
VAR_GLOBAL
  QUALITY_PB_NX1005_I:  LibDataTypes.QUALITY:= (VALIDITY:= VALIDITY_INVALID,
                          FLAGS:= (FLAG_RESTART:= TRUE));
  QUALITY_PB_NX1005_O:  LibDataTypes.QUALITY:= (VALIDITY:= VALIDITY_INVALID,
                          FLAGS:= (FLAG_RESTART:= TRUE));
  QUALITY_PB_NX6000:  LibDataTypes.QUALITY:= (VALIDITY:= VALIDITY_INVALID,
                        FLAGS:= (FLAG_RESTART:= TRUE));
  QUALITY_PB_NX6100:  LibDataTypes.QUALITY:= (VALIDITY:= VALIDITY_INVALID,
                        FLAGS:= (FLAG_RESTART:= TRUE));
END_VAR
```

5.2.5.1.6. *PROFIBUS Digital Inputs Quality*

```
// PROFIBUS digital input quality update, module NX1005

// In communication success case with PROFIBUS slave (address = 99) ...
IF DG_NX5001.tMstStatus.abySlv_State.bSlave_99 = TRUE THEN
  // Waits the PROFIBUS slave become apt to exchange data and diagnostics
  // (It is necessary to wait, avoiding invalid quality generation)
  IF  DG_NX5110.tPbusHeadA.tStatus1.bStation_Non_Existent = FALSE AND
    DG_NX5110.tPbusHeadA.tStatus1.bStation_Not_Ready = FALSE AND
    DG_NX5110.tPbusHeadA.wIdentNumber > 0 THEN
    QUALITY_PB_NX1005_I.FLAGS.FLAG_COMM_FAIL:= FALSE;
    // If there is a module present on the bus (slot = 2) and
    // if there is no modules config problem (general) and
    // if there is no config problem in that module (specific) and
    // if there is no fatal error identification by the module ...
    IF (DG_NX5110.tPbusHeadA.dwModuleNotPresent AND SHL(1, 2)) = 0 AND
      DG_NX5110.tPbusHeadA.tSummarized.bConfigMismatch = FALSE AND
      DG_NX1005_24_Vdc_8_DO_Trans_8_DI.tGeneral.bConfigMismatch = FALSE AND
      DG_NX1005_24_Vdc_8_DO_Trans_8_DI.tGeneral.bFatalError = FALSE THEN
      QUALITY_PB_NX1005_I.VALIDITY:= VALIDITY_GOOD;
      QUALITY_PB_NX1005_I.FLAGS.FLAG_RESTART:= FALSE;
      QUALITY_PB_NX1005_I.FLAGS.FLAG_FAILURE:= FALSE;
      QUALITY_PB_NX1005_I.FLAGS.FLAG_OLD_DATA:= FALSE;
    ELSE
      QUALITY_PB_NX1005_I.VALIDITY:= VALIDITY_INVALID;
      QUALITY_PB_NX1005_I.FLAGS.FLAG_FAILURE:= TRUE;
      // If the point have ever been updated once ...
      IF NOT QUALITY_PB_NX1005_I.FLAGS.FLAG_RESTART THEN
        QUALITY_PB_NX1005_I.FLAGS.FLAG_OLD_DATA:= TRUE;
      END_IF
```

```
    END_IF
  END_IF
// In PROFIBUS communication failure with the PROFIBUS slave ...
ELSE
  QUALITY_PB_NX1005_I.VALIDITY:= VALIDITY_INVALID;
  QUALITY_PB_NX1005_I.FLAGS.FLAG_COMM_FAIL:= TRUE;
  QUALITY_PB_NX1005_I.FLAGS.FLAG_FAILURE:= FALSE;
  // If the point have ever been updated once ...
  IF NOT QUALITY_PB_NX1005_I.FLAGS.FLAG_RESTART THEN
    QUALITY_PB_NX1005_I.FLAGS.FLAG_OLD_DATA:= TRUE;
  END_IF
END_IF
```

5.2.5.1.7. PROFIBUS Digital Output Quality

```
// PROFIBUS digital output quality update, module NX1005

// In communication success case with PROFIBUS slave (address = 99) ...
IF DG_NX5001.tMstStatus.abySlv_State.bSlave_99 = TRUE THEN
// Waits the PROFIBUS slave become apt to exchange data and diagnostics
// (It is necessary to wait, avoiding invalid quality generation)
IF  DG_NX5110.tPbusHeadA.tStatus1.bStation_Non_Existent = FALSE AND
  DG_NX5110.tPbusHeadA.tStatus1.bStation_Not_Ready = FALSE AND
  DG_NX5110.tPbusHeadA.wIdentNumber > 0 THEN
QUALITY_PB_NX1005_O.FLAGS.FLAG_COMM_FAIL:= FALSE;
// If there is a module present on the bus (slot = 2) and
// if there is no modules config problem (general) and
// if there is no config problem in that module (specific) and
// if there is no fatal error identification by the module and
// if there is no outputs short circuit indication and
// if there is no external power supply missing indication ...
IF (DG_NX5110.tPbusHeadA.dwModuleNotPresent AND SHL(1, 2)) = 0 AND
  DG_NX5110.tPbusHeadA.tSummarized.bConfigMismatch = FALSE AND
  DG_NX1005_24_Vdc_8_DO_Trans_8_DI.tGeneral.bConfigMismatch = FALSE AND
  DG_NX1005_24_Vdc_8_DO_Trans_8_DI.tGeneral.bFatalError = FALSE AND
  DG_NX1005_24_Vdc_8_DO_Trans_8_DI.tDetailed.bOutputShortCircuit = FALSE AND
  DG_NX1005_24_Vdc_8_DO_Trans_8_DI.tDetailed.bNoExternalSupply = FALSE THEN
    QUALITY_PB_NX1005_O.VALIDITY:= VALIDITY_GOOD;
    QUALITY_PB_NX1005_O.FLAGS.FLAG_RESTART:= FALSE;
    QUALITY_PB_NX1005_O.FLAGS.FLAG_FAILURE:= FALSE;
    QUALITY_PB_NX1005_O.FLAGS.FLAG_OLD_DATA:= FALSE;
ELSE
    QUALITY_PB_NX1005_O.VALIDITY:= VALIDITY_INVALID;
    QUALITY_PB_NX1005_O.FLAGS.FLAG_FAILURE:= TRUE;
    // If the point have ever been updated once ...
    IF NOT QUALITY_PB_NX1005_O.FLAGS.FLAG_RESTART THEN
      QUALITY_PB_NX1005_O.FLAGS.FLAG_OLD_DATA:= TRUE;
    END_IF
END_IF
END_IF
// In PROFIBUS communication failure with the PROFIBUS slave ...
ELSE
```

```
QUALITY_PB_NX1005_O.VALIDITY:= VALIDITY_INVALID;
QUALITY_PB_NX1005_O.FLAGS.FLAG_COMM_FAIL:= TRUE;
QUALITY_PB_NX1005_O.FLAGS.FLAG_FAILURE:= FALSE;
// If the point have ever been updated once ...
IF NOT QUALITY_PB_NX1005_O.FLAGS.FLAG_RESTART THEN
  QUALITY_PB_NX1005_O.FLAGS.FLAG_OLD_DATA:= TRUE;
END_IF
END_IF
```

5.2.5.1.8. *PROFIBUS Analog Inputs Quality*

```
// PROFIBUS analog input quality update, module NX6000

// In communication success case with PROFIBUS slave (address = 99) ...
IF DG_NX5001.tMstStatus.abySlv_State.bSlave_99 = TRUE THEN
// Waits the PROFIBUS slave become apt to exchange data and diagnostics
// (It is necessary to wait, avoiding invalid quality generation)
IF  DG_NX5110.tPbusHeadA.tStatus1.bStation_Non_Existent = FALSE AND
  DG_NX5110.tPbusHeadA.tStatus1.bStation_Not_Ready = FALSE AND
  DG_NX5110.tPbusHeadA.wIdentNumber > 0 THEN
QUALITY_PB_NX6000.FLAGS.FLAG_COMM_FAIL:= FALSE;
// If there is a module present on the bus (slot = 3) and
// if there is no modules config problem (general) and
// if there is no config problem in that module (specific) and
// if there is no fatal error identification by the module and
// if there is no calibration error indication and
// if there is no over/under range error indication and
// if there is no error indication of input in open loop ...
IF (DG_NX5110.tPbusHeadA.dwModuleNotPresent AND SHL(1, 3)) = 0 AND
  DG_NX5110.tPbusHeadA.tSummarized.bConfigMismatch = FALSE AND
  DG_NX6000_8_AI_Voltage_Current.tGeneral.bConfigMismatch = FALSE AND
  DG_NX6000_8_AI_Voltage_Current.tGeneral.bFatalError = FALSE AND
  DG_NX6000_8_AI_Voltage_Current.tGeneral.bCalibrationError = FALSE AND
  DG_NX6000_8_AI_Voltage_Current.tDetailed.tAnalogInput_00.bOverRange = FALSE
   AND
  DG_NX6000_8_AI_Voltage_Current.tDetailed.tAnalogInput_00.bUnderRange = FALSE
   AND
  DG_NX6000_8_AI_Voltage_Current.tDetailed.tAnalogInput_00.bOpenLoop = FALSE
   THEN
    QUALITY_PB_NX6000.VALIDITY:= VALIDITY_GOOD;
    QUALITY_PB_NX6000.FLAGS.FLAG_RESTART:= FALSE;
    QUALITY_PB_NX6000.FLAGS.FLAG_FAILURE:= FALSE;
    QUALITY_PB_NX6000.FLAGS.FLAG_OLD_DATA:= FALSE;
    QUALITY_PB_NX6000.FLAGS.FLAG_INACCURATE:= FALSE;
    QUALITY_PB_NX6000.FLAGS.FLAG_OUT_OF_RANGE:= FALSE;
ELSE
    // Condition to turns on imprecision indication
    // (check first, because invalid validity must prevail)
    IF DG_NX6000_8_AI_Voltage_Current.tGeneral.bCalibrationError = TRUE THEN
        QUALITY_PB_NX6000.VALIDITY:= VALIDITY_QUESTIONABLE;
        QUALITY_PB_NX6000.FLAGS.FLAG_INACCURATE:= TRUE;
    ELSE
```

```
          QUALITY_PB_NX6000.FLAGS.FLAG_INACCURATE:= FALSE;
      END_IF
      // Condition to turns on out of range indication
      // (check first, because invalid validity must prevail)
      IF  DG_NX6000_8_AI_Voltage_Current.tDetailed.tAnalogInput_00.bOverRange =
     TRUE OR
        DG_NX6000_8_AI_Voltage_Current.tDetailed.tAnalogInput_00.bUnderRange =
     TRUE THEN
          QUALITY_PB_NX6000.VALIDITY:= VALIDITY_QUESTIONABLE;
          QUALITY_PB_NX6000.FLAGS.FLAG_OUT_OF_RANGE:= TRUE;
      ELSE
          QUALITY_PB_NX6000.FLAGS.FLAG_OUT_OF_RANGE:= FALSE;
      END_IF
      // Condition to turns on general failure indication (priority)
      IF (DG_NX5110.tPbusHeadA.dwModuleNotPresent AND SHL(1, 3)) > 0 OR
        DG_NX5110.tPbusHeadA.tSummarized.bConfigMismatch = TRUE OR
        DG_NX6000_8_AI_Voltage_Current.tGeneral.bConfigMismatch = TRUE OR
        DG_NX6000_8_AI_Voltage_Current.tGeneral.bFatalError = TRUE OR
        DG_NX6000_8_AI_Voltage_Current.tDetailed.tAnalogInput_00.bOpenLoop = TRUE
     THEN
          QUALITY_PB_NX6000.VALIDITY:= VALIDITY_INVALID;
          QUALITY_PB_NX6000.FLAGS.FLAG_FAILURE:= TRUE;
          // If the point have ever been updated once ...
          IF  NOT QUALITY_PB_NX6000.FLAGS.FLAG_RESTART AND
          NOT DG_NX6000_8_AI_Voltage_Current.tDetailed.tAnalogInput_00.bOpenLoop
     THEN
              QUALITY_PB_NX6000.FLAGS.FLAG_OLD_DATA:= TRUE;
          END_IF
      ELSE
          QUALITY_PB_NX6000.FLAGS.FLAG_RESTART:= FALSE;
          QUALITY_PB_NX6000.FLAGS.FLAG_FAILURE:= FALSE;
          QUALITY_PB_NX6000.FLAGS.FLAG_OLD_DATA:= FALSE;
      END_IF
    END_IF
  END_IF
  // In PROFIBUS communication failure with the PROFIBUS slave ...
  ELSE
  QUALITY_PB_NX6000.VALIDITY:= VALIDITY_INVALID;
  QUALITY_PB_NX6000.FLAGS.FLAG_COMM_FAIL:= TRUE;
  QUALITY_PB_NX6000.FLAGS.FLAG_FAILURE:= FALSE;
  // If the point have ever been updated once ...
  IF  NOT QUALITY_PB_NX6000.FLAGS.FLAG_RESTART AND
    NOT DG_NX6000_8_AI_Voltage_Current.tDetailed.tAnalogInput_00.bOpenLoop THEN
    QUALITY_PB_NX6000.FLAGS.FLAG_OLD_DATA:= TRUE;
  END_IF
  END_IF
```

5.2.5.1.9. PROFIBUS Analog Output Quality

```
// PROFIBUS analog output quality update, module NX6100

// In communication success case with PROFIBUS slave (address = 99) ...
IF DG_NX5001.tMstStatus.abySlv_State.bSlave_99 = TRUE THEN
// Waits the PROFIBUS slave become apt to exchange data and diagnostics
// (It is necessary to wait, avoiding invalid quality generation)
IF  DG_NX5110.tPbusHeadA.tStatus1.bStation_Non_Existent = FALSE AND
  DG_NX5110.tPbusHeadA.tStatus1.bStation_Not_Ready = FALSE AND
  DG_NX5110.tPbusHeadA.wIdentNumber > 0 THEN
QUALITY_PB_NX6100.FLAGS.FLAG_COMM_FAIL:= FALSE;
// If there is a module present on the bus (slot = 4) and
// if there is no modules config problem (general) and
// if there is no config problem in that module (specific) and
// if there is no fatal error identification by the module and
// if there is no calibration error indication and
// if there is no external power supply missing indication and
// if there is no error indication of output in open loop and
// if there is no outputs short circuit indication ...
IF (DG_NX5110.tPbusHeadA.dwModuleNotPresent AND SHL(1, 4)) = 0 AND
  DG_NX5110.tPbusHeadA.tSummarized.bConfigMismatch = FALSE AND
  DG_NX6100_4_AO_Voltage_Current.tGeneral.bConfigMismatch = FALSE AND
  DG_NX6100_4_AO_Voltage_Current.tGeneral.bFatalError = FALSE AND
  DG_NX6100_4_AO_Voltage_Current.tGeneral.bCalibrationError = FALSE AND
  DG_NX6100_4_AO_Voltage_Current.tGeneral.bNoExternalSupply = FALSE AND
  DG_NX6100_4_AO_Voltage_Current.tDetailed.tAnalogOutput_00.bOpenLoop = FALSE
   AND
  DG_NX6100_4_AO_Voltage_Current.tDetailed.tAnalogOutput_00.bShortCircuit =
   FALSE THEN
    QUALITY_PB_NX6100.VALIDITY:= VALIDITY_GOOD;
    QUALITY_PB_NX6100.FLAGS.FLAG_RESTART:= FALSE;
    QUALITY_PB_NX6100.FLAGS.FLAG_FAILURE:= FALSE;
    QUALITY_PB_NX6100.FLAGS.FLAG_INACCURATE:= FALSE;
    QUALITY_PB_NX6100.FLAGS.FLAG_OLD_DATA:= FALSE;
  ELSE
    // Condition to turns on imprecision indication
    // (check first, because invalid validity must prevail)
    IF DG_NX6100_4_AO_Voltage_Current.tGeneral.bCalibrationError = TRUE THEN
        QUALITY_PB_NX6100.VALIDITY:= VALIDITY_QUESTIONABLE;
        QUALITY_PB_NX6100.FLAGS.FLAG_INACCURATE:= TRUE;
    ELSE
        QUALITY_PB_NX6100.FLAGS.FLAG_INACCURATE:= FALSE;
    END_IF
    // Condition to turns on general failure indication (priority)
    IF (DG_NX5110.tPbusHeadA.dwModuleNotPresent AND SHL(1, 4)) > 0 OR
      DG_NX5110.tPbusHeadA.tSummarized.bConfigMismatch = TRUE OR
      DG_NX6100_4_AO_Voltage_Current.tGeneral.bConfigMismatch = TRUE OR
      DG_NX6100_4_AO_Voltage_Current.tGeneral.bFatalError = TRUE OR
      DG_NX6100_4_AO_Voltage_Current.tGeneral.bNoExternalSupply = TRUE OR
      DG_NX6100_4_AO_Voltage_Current.tDetailed.tAnalogOutput_00.bOpenLoop = TRUE
     OR
      DG_NX6100_4_AO_Voltage_Current.tDetailed.tAnalogOutput_00.bShortCircuit =
    TRUE THEN
        QUALITY_PB_NX6100.VALIDITY:= VALIDITY_INVALID;
```

```
          QUALITY_PB_NX6100.FLAGS.FLAG_FAILURE:= TRUE;
          // If the point have ever been updated once ...
          IF  NOT QUALITY_PB_NX6100.FLAGS.FLAG_RESTART AND NOT
            DG_NX6100_4_AO_Voltage_Current.tDetailed.tAnalogOutput_00.bOpenLoop
     THEN
              QUALITY_PB_NX6100.FLAGS.FLAG_OLD_DATA:= TRUE;
          END_IF
      ELSE
          QUALITY_PB_NX6100.FLAGS.FLAG_RESTART:= FALSE;
          QUALITY_PB_NX6100.FLAGS.FLAG_FAILURE:= FALSE;
          QUALITY_PB_NX6100.FLAGS.FLAG_OLD_DATA:= FALSE;
      END_IF
  END_IF
  END_IF
  // In PROFIBUS communication failure with the PROFIBUS slave ...
  ELSE
  QUALITY_PB_NX6100.VALIDITY:= VALIDITY_INVALID;
  QUALITY_PB_NX6100.FLAGS.FLAG_COMM_FAIL:= TRUE;
  QUALITY_PB_NX6100.FLAGS.FLAG_FAILURE:= FALSE;
  // If the point have ever been updated once ...
  IF  NOT QUALITY_PB_NX6100.FLAGS.FLAG_RESTART AND
    NOT DG_NX6100_4_AO_Voltage_Current.tDetailed.tAnalogOutput_00.bOpenLoop THEN
  QUALITY_PB_NX6100.FLAGS.FLAG_OLD_DATA:= TRUE;
  END_IF
  END_IF
```

5.3. Serial Interfaces Configuration

5.3.1. COM 1

The COM 1 communication interface, is composed by a DB9 female connector for RS-232C standard. It allows the point to point communication (or in network by using a converter) in MODBUS RTU slave or MODBUS RTU master open protocols.

The parameters which must be configured for the proper functioning of the application are described below.

When using the MODBUS master/slave protocol, some of these parameters (such as Serial Mode, Data Bits, RX Threshold and Serial Events) are automatically adjusted by MasterTool for the correct operation of this protocol.

Configuration	Description	Default	Options
Serial Type	Serial channel type configuration.	RS-232C	RS-232C
Baud Rate	Serial communication port speed configuration.	115200	200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps
Parity	Serial port parity configuration.	None	Odd Even Space Mark None
Data Bits	Sets the serial communication character bits quantity.	8	5, 6, 7 and 8
Stop Bits	Sets the serial port stop bits.	1	1, 1.5 and 2

Configuration	Description	Default	Options
Serial Mode	Sets the serial port operation mode configuration.	Normal Mode	- Extended Mode: Extended operation mode which delivers information regarding the received data frame. - Normal Mode: Serial communication normal operation mode.

Table 58: RS-232 Standard Serial Configuration

Notes:

Extended Mode: This serial communication operation mode provides information regarding the data frame received. The information available is the following:

- One byte for the received data (RX_CHAR : BYTE): Store the five, six, seven or eight bits from the data received, depending on the serial communication configuration.
- One byte for the signal errors (RX_ERROR : BYTE): It has the format described below:
  - Bit 0: 0 - the character in bits 0 to 7 is valid. 1 - the character in bits 0 to 7 is not valid (or it cannot be valid), due to problems indicated in bits 10 to 15.
  - Bit 1: Not used.
  - Bit 2: Not used.
  - Bit 3: UART interruption error. The serial input remained in logic 0 (space) for a time greater than a character (start bit + data bits + parity bit + stop bits).
  - Bit 4: UART frame error. The logic 0 (space) was read when the first stop bit was expected and it should be logic 1 (mark).
  - Bit 5: UART parity error. The parity bit read is not correct according to the calculated one.
  - Bit 6: UART overrun error. Data was lost during the FIFO UART reading. New characters were received before the later ones were removed. This error will only be indicated in the first character read after the overrun error indication. This means some old data were lost.
  - Bit 7: RX line overrun error. This character was written when the RX line was completed, overwriting the unread characters.
- Two bytes for the timestamp signal (RX_TIMESTAMP : WORD): Indicates the silence time, within the 0 to 65535 interval, using 10 us as base. It saturates in 655.35 ms if the silence time is higher than 65535 units. The RX_TIMESTAMP of a character measures the time from a reference which can be any of the three options below:
  - On most of the cases, the end of the later character.
  - Serial port configuration.
  - The end of serial communication using the SERIAL_TX FB, in other words, when the last character is sent on line.

  Besides measuring the silence between characters, the RX_TIMESTAMP is also important as it measures the silence time of the last character on the RX line. The silence measuring is important for the correct protocol implementation, as MODBUS RTU, for example. This protocol specifies an inter-frame greater than 3.5 characters and an inter-byte less than 1.5 characters.

Data Bits: The serial interfaces *Data Bits* configuration limits the *Stop Bits* and Communication *Parity* fields. Therefore, the stop bits number and the parity method will vary according to the data bits number.

Data Bits	Stop Bits	Parity
5	1, 1.5	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO
6	1, 2	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO
7	1, 2	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO
8	1, 2	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO

Table 59: Specific Configurations

5.3.1.1. Advanced Configurations

The advanced configurations are related to the serial communication control, in other words, when it is necessary the utilization of a more accurate data transmission and reception control.

Configuration	Description	Default	Options
Advanced Port Parameters			
Handshake	Executes the request control for a command transmission through RS-232C interface.	RTS Off	- RTS: Enabled at the beginning of transmission and restarted, as fast as possible after the end of it. E.g. The RS-232/RS-485 external converter control. - RTS Off: Always disabled. - RTS On: Always enabled. - RTS/CTS: In case the CTS is disabled, the RTS is enabled. Therefore the CTS enabling must be waited until the transmission can start again and the RTS restarted, as fast as possible, at the end of transmission. E.g. the radio modems control using the same modem signal. - Manual RTS: the user is responsible for all control signals.
UART RX Threshold	Bytes quantity which must be received to generate a new UART interruption. Low values make the TIMESTAMP more precise when the EXTENDED MODE is used and minimizes the overrun errors. However, values too low may cause several interruptions delaying the CPU.	8	1, 4, 8 and 14

Configuration	Description	Default	Options
Serial Events			
RX on TX	When true, all received bytes during transmission will be discharged instead of going to the RX line. Used to disable the full-duplex operation of the RS-232C interface.	Disabled	- Enabled: Configuration enabled - Disabled: Configuration disabled
RX DCD Event	When true, generates an external event due to DCD signal change.	Enabled	- Enabled: Configuration enabled - Disabled: Configuration disabled
RX CTS Event	When true, generates an external event due to CTS signal change.	Enabled	- Enabled: Configuration enabled - Disabled: Configuration disabled

Table 60: RS-232 Standard Serial Advanced Configurations

Notes:

RX on TX: This advanced parameter is valid for RS-232C settings and RS-422.

RX DCD Event: External events such as the DCD signal COM 1 of the CPUs NX3010, NX3020, NX3030, may be associated only to tasks of custom project profile, for further information, please see the MasterTool IEC XE User Manual – MU299609.

RX CTS Event: External events such as the CTS signal COM 1 of the CPUs NX3010, NX3020, NX3030, may be associated only to tasks of custom project profile, for further information, please see the MasterTool IEC XE User Manual – MU299609.

5.3.2. COM 2

The serial interfaces *Data Bits* configuration limits the *Stop Bits* and Communication *Parity* fields. Therefore, the number of stop bits and the parity method will vary according to the data bits number.

The table below shows the allowed configurations interfaces.

Data Bits	Stop Bits	Parity
5	1, 1.5	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO
6	1, 2	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO
7	1, 2	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO
8	1, 2	NO PARITY, ODD, EVEN, PARITY ALWAYS ONE, PARITY ALWAYS ZERO

Table 61: Specific Configurations

5.3.2.1. Advanced Configurations

The advanced configurations are related to the serial communication control, in other words, when it is necessary the utilization of a more accurate data transmission and reception control.

Configuration	Description	Default	Options
UART RX Threshold	Bytes quantity which must be received for a new UART interruption to be generated. Low values make the TIMESTAMP more precise when the EXTENDED MODE is used and minimizes the overrun errors. However, values too low may cause several interruptions delaying the CPU.	8	1, 4, 8 and 14

Table 62: RS-485/RS-422 Standard Serial Advanced Configurations

5.4. Ethernet Interfaces Configuration

Nexto CPUs can provide more local Ethernet interfaces. The NX3030 CPU has NET 1 and NET 2. In addition of the local Ethernet interfaces, the Nexto Series also provides remote Ethernet interfaces through the inclusion of the module NX5000. NX5000 modules have only the NET 1 interface.

5.4.1. Internal Ethernet Interfaces

The interfaces are composed by a RJ45 communication connector 10/100Base-TX standard. It allows the point to point or network communication in the following open protocols, for example: MODBUS TCP Client, MODBUS RTU via TCP Client, MODBUS TCP Server and MODBUS RTU via TCP Server.

The parameters which must be configured for the proper functioning of the application are described below.

5.4.1.1. NET 1

Configuration	Description	Default	Options
Obtain an IP address automatically	Enables the DHCP Client functionality on the device for automatic IP assignment	Unmarked	Marked or Unmarked
IP Address	IP address of the controller in the Ethernet bus	192.168.15.1	1.0.0.1 to 223.255.255.254
Subnetwork Mask	Subnet mask of the controller in the Ethernet bus	255.255.255.0	128.0.0.0 to 255.255.255.252
Gateway Address	Controller Gateway address in the Ethernet bus	192.168.15.253	0.0.0.0 to 223.255.255.254

Table 63: NET 1 Configuration

5.4.1.2. NET 2

Configuration	Description	Default	Options
IP Address	IP address of the controller in the Ethernet bus	192.168.16.1	1.0.0.1 to 223.255.255.254
Subnetwork Mask	Subnet mask of the controller in the Ethernet bus	255.255.255.0	128.0.0.0 to 255.255.255.252
Gateway Address	Gateway address of the controller in the Ethernet bus	192.168.16.253	0.0.0.0 to 223.255.255.254

Table 64: NET 2 Configuration

ATTENTION

It is not possible to configure more than one Ethernet interface of a CPU on the same subnet, and this type of configuration is blocked by the MasterTool tool. Therefore, each Ethernet interface must be configured on a different subnet.

5.4.2. NX5000 Remote Ethernet Interfaces

5.4.2.1. NET 1

The interface is composed by a RJ45 communication connector 10/100Base-TX standard. It allows the point to point or network communication in the following open protocols: MODBUS TCP Client, MODBUS RTU via TCP Client, MODBUS TCP Server and MODBUS RTU via TCP Server.

The parameters which must be configured for the proper functioning of the application are described below.

Configuration	Description	Default	Options
IP Address	IP address of the controller in the Ethernet bus	192.168.xx.68	1.0.0.1 to 223.255.255.254
Subnetwork Mask	Subnet mask of the controller in the Ethernet bus	255.255.255.0	128.0.0.0 to 255.255.255.252
Gateway Address	Gateway address of the controller in the Ethernet bus	192.168.xx.253	0.0.0.0 to 223.255.255.254

Table 65: NX5000 Remote NET 1 Configuration

5.4.2.2. Operation Mode of the NX5000 Remote Ethernet Interface

The NX5000 modules can be inserted in the project to increase the number of Ethernet interfaces if the local CPU interfaces are not enough.

The Ethernet channels of the NX5000 modules can be used individually, or arranged in redundant pairs.

5.4.2.2.1. Redundant Mode

A pair of two Ethernet ports forming a redundant pair has a single IP address tied to the port pair. In this way, a client, such as SCADA or MasterTool, connected to a server at the CPU, does not have to worry about changing the IP address if some of the ports in the redundant pair fail.

In order to put together two NX5000 modules as a redundant pair, these two modules must necessary occupy adjacent positions on the backplane rack and the checkbox *Redundant* from the module on the left must be selected, as show in the figure below. By doing this, the parameters edition of the module on the right is blocked. The parameters edited in the module inserted on the left get common for the two modules.

On the other hand, clearing the *Redundant* checkbox from the module on the left causes the separation of the modules, which return to behave as individual modules without redundant.

When the *Redundant* Mode is selected, on the same screen other parameters are automatically enabled and must be configured:

- **Period of Redundancy Test (ms):** Period for sending the communication test frame between the two NETs. Can be configured with values between 100 and 9900, default 500
- **Retries of Redundancy Test:** Maximum number of times the NET that sent the frame will wait for a response. Can be configured to values between 1 and 100, default 4
- **Switching Period (s):** Maximum time that NET Active will wait for any given packet. Can be configured with values between 1 and 25, default 10

If the response time of the *Redundancy Test* reaches *Test Period* times the *Number of Retries* and the active interface remains longer than the *Switching Period* without receiving any packets, a switchover occurs, making the previously inactive interface active. It is important to note that have a delay between fault detection and activation of the inactive interface due to the time required for its configuration. This delay can be up to a few tens of milliseconds.

When one of the NETs is active, it will assume the configured IP address, and the inactive NET will remain with its IP Address, Subnet Mask and Gateway Address parameters blank in the CPU diagnostics.

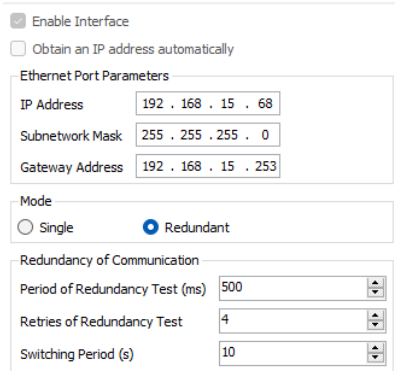


Figure 51: Advanced Configuration of Remote Ethernet Interface - NX5000

5.4.3. Reserved TCP/UDP Ports

The following TCP/UDP ports of the Ethernet interfaces, both local and remote, are used by CPU services (depending on availability according to table Protocols) and, therefore, are reserved and must not be used by the user.

Service	TCP	UDP
System Web Page	80	-
SNTP	-	123
SNMP	-	161
MODBUS TCP	502*	-
MasterTool MT8500	1217*	1740:1743
SQL Server	1433	-
MQTT	1883* / 8883*	-
EtherNet/IP	44818	2222
IEC 60870-5-104	2404*	-
OPC UA	4840	-
WEBVISU	8080	-
CODESYS ARTI	11740	-
PROFINET	-	34964
Portainer Docker	9000	-

Table 66: Reserved TCP/UDP ports

* Default port, but user changeable.

5.5. Protocols Configuration

Independently of the protocols used in each application, the Nexto Series CPUs has some maximum limits for each CPU model. There are basically two different types of communication protocols: symbolic and direct representation mappings. The maximum limit of mappings as well as the maximum protocol quantity (instances) is defined on table below:

	NX3030
Mapped Points	20480
Mappings (Per Instance / Total)	5120 / 20480
Requests	512
NETs – Client or Server Instances (Per NET / Total)	4 / 16
COM (n) – Master or Slave Instances	1
Control Centers	3

Table 67: Protocols Limits per CPU

Notes:

Mapped Points: It refers to the maximum number of mapped points supported by the CPU. Each mapping supports one or more mapped points, depending on the size of the data when used with variables of type ARRAY.

Mappings: A "mapping" is the relationship between an internal application variable and an object of the application protocol. This field informs the maximum number of mappings supported by the CPU. It corresponds to the sum of all mappings made within the instances of communication protocols and their respective devices.

Requests: The sum of the requests of the communications protocols, declared on devices, may not exceed the maximum number of requests supported by the CPU.

NETs – Clients or Servers Instances: This field defines the maximum number of protocol instances per Ethernet interface, and also the total maximum distributed along all the Ethernet interfaces of the system.

COM (n) – Master or Slave Instances: Due to its characteristics, each serial interface supports only one communication protocol instance. Examples of instances compatible with serial interfaces: MODBUS RTU Master and MODBUS RTU Slave.

Control Centers: "Control Center" is all client device connected to the CPU through protocol IEC 60870-5-104. This field informs the maximum of client devices of control center type supported by the CPU. Correspond to the sum of all client devices of communication protocol IEC 60870-5-104 Server (does not include master or clients from MODBUS RTU Slave, MODBUS Server and DNP3 Server protocols).

The limitations of the MODBUS protocol for Direct Representation and symbolic mapping for the CPUs can be seen in Tables 68 and 69, respectively.

Limitations	MODBUS RTU Master	MODBUS RTU Slave	MODBUS Ethernet Client	MODBUS Ethernet Server
Mappings per instance	128	32	128	32
Devices per instance	64	1(1)	64	64(2)
Mappings per device	32	32	32	32
Simultaneous requests per instance	-	-	128	64
Simultaneous requests per device	-	-	8	64

Table 68: MODBUS Protocol Limitations for Direct Representation

Notes:

Devices per instance:

- Master or Client Protocols: number of slaves or server devices supported by each Master or Slave protocol instance.
- MODBUS RTU Slave Protocol: the limit (1) informed relates to serial interfaces that do not allow a Slave to establish communication through the same serial interface, simultaneously, with more than one Master device. It's not necessary, nor is it possible to declare or configure the Master device in the instance of the MODBUS RTU slave protocol. The master device will have access to all the mappings made directly on the instance of MODBUS RTU slave protocol.

- MODBUS RTU Server Protocol: the limit [(2)] informed relates to the Ethernet interfaces, which limit the number of connections that can be established with other devices through a single Ethernet interface. It is not necessary, nor is it possible to declare or configure Clients devices in the instance of the MODBUS Server protocol. All Clients devices will have access to all the mappings made directly in the instance of the MODBUS Server protocol.

Mappings per device: The maximum number of mappings per device, despite being listed above, is also limited by the protocol maximum number of mappings. Also to be considered the maximum CPU mappings as in Table 67.

Simultaneous Requests per Instance: Number of requests that can be simultaneously transmitted by each Client protocol instance or that can be received simultaneously by each Server protocol instance. MODBUS RTU protocol instances, Master or Slave, do not support simultaneous requests.

Simultaneous Requests per Device: Number of requests that can be simultaneously transmitted to each MODBUS Server device, or may be received simultaneously by each MODBUS client device. MODBUS RTU devices, Master or Slave, do not support simultaneous requests.

Limitations	MODBUS RTU Master	MODBUS RTU Slave	MODBUS Ethernet Client	MODBUS Ethernet Server
Devices per instance	64	1[(1)]	64	64[(2)]
Requests per device	32	-	32	-
Simultaneous requests per instance	-	-	128	64
Simultaneous requests per device	-	-	8	64

Table 69: MODBUS Protocol Limitations for Symbolic Mappings

Notes:

Devices per instance:

- Master or Client Protocol: Number of slave or server devices supported by each Master or Client protocol instance.
- MODBUS RTU Slave Protocol: the limit [(1)] informed relates to serial interfaces that do not allow a Slave to establish communication through the same serial interface, simultaneously, with more than one Master device. It's not necessary, nor is it possible to declare or configure the Master device in the instance of the MODBUS RTU slave protocol. The master device will have access to all the mappings made directly on the instance of MODBUS RTU slave protocol.
- MODBUS RTU Server Protocol: the limit [(2)] informed relates to the Ethernet interfaces, which limit the amount of connections that can be established with other devices through a single Ethernet interface. It is not necessary, nor is it possible to declare or configure Clients devices in the instance of the MODBUS Server protocol. All Clients devices will have access to all the mappings made directly in the instance of the MODBUS Server protocol.

Requests by device: Number of requests, such as reading or writing holding registers, which can be configured for each of the devices (slaves or servers) from Master or Client protocols instances. This parameter does not apply to instances of Slave or Server protocols.

Simultaneous Requests per Instance: Number of requests that can be simultaneously transmitted by each client protocol instance or that can be received simultaneously by each server protocol instance. MODBUS RTU protocol instances, Master or Slave, do not support simultaneous requests.

Simultaneous Requests per Device: Number of requests that can be simultaneously transmitted for each MODBUS server device, or may be received simultaneously from each MODBUS client device. MODBUS RTU devices, Master or Slave do not support simultaneous requests.

ATTENTION

Simultaneous requests to a variable associated to communication points, those which support the SBO operation mode (Select Before Operate), even being received from different devices are not supported. Once started the selection/operation of a point by a specific device, that must be finished before this point become able to be commanded by another device.

The protocol IEC 60870-5-104 Server limitations can be watched on table below.

Limitations	IEC 60870-5-104 Server
Devices per instance	3
Simultaneous requests per instance	3
Simultaneous requests per device	1

Table 70: Protocol IEC 60870-5-104 Server Limits

Notes:

Devices per instance: Quantity of client devices, of type control center, supported for each IEC 60870-5-104 Server protocol instance. The limit informed might be smaller because of the CPU total limits (check Table 67).

Simultaneous requests per instance: Quantity of requests that can be received simultaneously by each instance of Server protocol.

Simultaneous requests per device: Quantity of requests that can be received simultaneously of each IEC 60870-5-104 Client device.

5.5.1. Protocol Behavior x CPU State

The table below shows in detail the behavior of each configurable protocol in Nexto Series CPUs in every state of operation.

Protocol	Type	CPU operational state					
		STOP			RUN		
		After download, before application starts	After the application goes to STOP (PAUSE)	After an exception	Non redundant or Active	Redundant in Stand-by	After a breakpoint in MainPrg
MODBUS Symbol	Slave/Server	✓	✓	✓	✓	✓	✓
	Master/Client	✗	✗	✗	✓	✓	✓
MODBUS	Slave/Server	✗	✗	✗	✓	✓	✗
	Master/Client	✗	✗	✗	✓	✓	✓
SOE (DNP3)	Outstation	✓	✓	✓	✓	✗	✓
IEC 60870-5-104	Server	✗	✗	✗	✓	✗	✓
EtherCAT	Master	✓	✓	✗	✓	NA	✓
OPC DA	Server	✓	✓	✓	✓	✗	✓
OPC UA	Server	✓	✓	✓	✓	✓	✓
SNTP	Client	✓	✓	✓	✓	✓	✓
HTTP	Server	✓	✓	✓	✓	✓	✓
SNMP	Agent	✓	✓	✓	✓	✓	✓
EtherNet/IP	Scanner	✓	✓	✗	✓	NA	✗
	Adapter	✗	✓	✗	✓	NA	✗

Table 71: Protocol Behavior x CPU State

Notes:

Symbol ✓: Protocol remains active and operating normally.

Symbol ✗: Protocol is disabled.

MODBUS Symbol Slave/Server: To keep the protocol communicating when the CPU isn't in RUN or after a breakpoint, it's need to check the option "*Keep the communication running on CPU stop*".

5.5.2. Double Points

The input and output double digital points representation is done through a special data type called DBP (defined in the *LibDataTypes* library). This type consist basically in a structure of two BOOL type elements, called OFF and ON (equivalent to TRIP and CLOSE respectively).

In Nexto, variables of this type cannot be associated to digital input and output modules, being necessary the single digital points mapping, BOOL type, and the treatment by application to conversion in double points.

To further information about the double points mapping in the input and output digital modules check the IEC 60870-5-104 Server section.

5.5.3. CPU's Events Queue

The CPU owns an events queue of type FIFO (First In, First Out) used to store temporarily the events related to communication points until they are moved to their final destiny.

All communication points events generated in the CPU are directed and stored in CPU's queue. This queue has the following features:

- Size: 1000 events
- Retentivity : it is not retentive
- Overflow policy: keep the newests

ATTENTION

In the Nexto PLC, the events queue is stored in a non-retentive memory area (volatile). This way, the events present in CPU's queue, which haven't been transmitted yet to the control center, are going to be lost in a CPU's eventual power off.

The CPU's event queue is redundant, that means it is synchronized each cycle between both CPUs, when is used CPU's redundancy. Further information can be found on the section about CPU redundancy.

The in and out of events in this queue follows the concept of producer/consumer. Producers are those system elements capable of generate events, adding events in the CPU's queue, while the consumers are those system elements which receive and use this events, taking them of the CPU's queue. The figure below describes this working, including the example of some events consumers and producers.

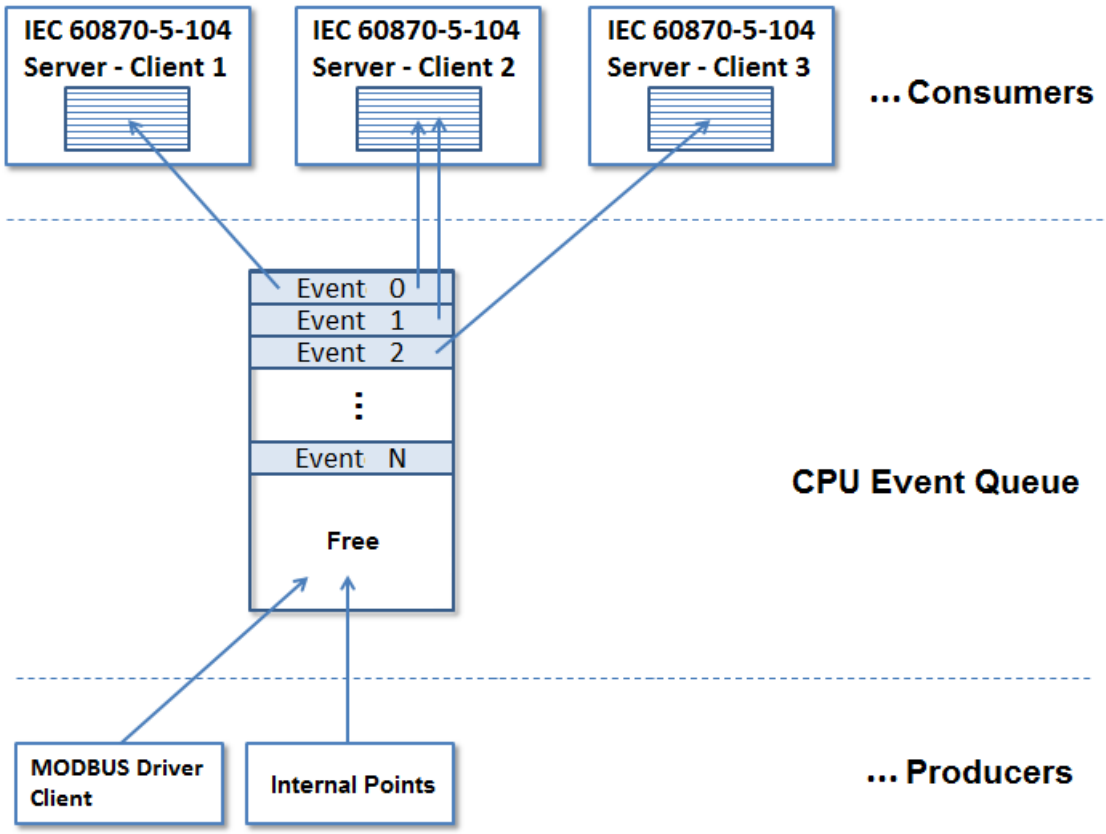


Figure 52: CPU's Event Queue

5.5.3.1. Consumers

The consumers are typically communication drivers that will communicate with SCADA or HMI. After been stored in CPU's queue, the consumers receive the events related to communication points mapped in its configuration. These events are then stored in a consumer's own events queue, which the size and working are described on the communication driver specific section.

5.5.3.2. Queue Functioning Principles

Once stored in CPU's queue, each event is transmitted to the consumer that has this communication point in its data base. On the figure above, the Event 0 is referred to a communication point mapped to two control centers IEC 60870-5-104 (Client 1 and 2). Thus the Event 1 is referred to a communication point mapped only to one control center IEC 60870-5-104 (Client 2). By its time, the Event 2 is referred to a communication point mapped to another control center IEC 60870-5-104 (Client 3).

The events remain stored in the CPU's queue until all its consumers acknowledge its receiving. The criteria used to confirm the receive is specific of each consumer. In case of the IEC 60870-5-104 Server, the acknowledge occurs when the event is transmitted to the IEC 60870-5-104 client.

In Nexto Series case, there are no diagnostics available to watch the CPU's events queue occupation, not even information about the queue overflow. However the consumers have a diagnostics group referred to its events queue. Further information can be found at the specific driver communication section.

5.5.3.2.1. Overflow Sign

The overflow sign to the consumers' events queue occurs in two situations:

- When the consumer events queue is out of space to store new events
- If the CPU aborted the event generation (because occurred to more events in a single execution cycle than the events queue total size)

5.5.3.3. Producers

The producers are typically communication drivers or PLC internal elements that are capable to generate events. The previous figure show some examples.

- **Internal Points:** This is a PLC's firmware internal element, which detects events each execution cycle (MainTask) to those communication points that don't have a defined origin and then inserts the events in the CPU's queue. The maximum number of events that can be detected in each MainTask cycle is equal to the CPU's events queue size. In case the number of generated events is bigger than this, in a single cycle, the exceeding are going to be lost.
- **MODBUS Driver (Client/Server/Master/Slave):** The variables value change caused by MODBUS read/write are detected at each MainTask cycle and then the events are inserted in CPU's queue. In Client/Master cases, are also generated quality events when there is a communication failure with the slave device.

5.5.4. Interception of Commands Coming from the Control Center

The Nexto PLC has a function block that allows selection commands and operation to the output points received by server drivers (IEC 60870-5-104 Server) been treated by the user logic. This resource allows the interlocking implementation, as well as the handling of the received command data in the user logic, or yet the command redirecting to different IEDs.

The commands interception is implemented by the *CommandReceiver* function block, defined in the *LibRtuStandard*. The input and output parameters are described on the following tables:

Parameter	Type	Description
bExec	BOOL	When TRUE, executes the command interception
bDone	BOOL	Indicates that the command output data have been already processed, releasing the function block to receive another command
dwVariableAddr	DWORD	Variable address, mapped in the server driver, which will receive the client command
eCommandResult	ENUM	Input action defined by user from the following list: SUCCESS(0) NOT_SUPPORTED(1) BLOCKED_BY_SWITCHING_HIERARCHY(2) SELECT_FAILED(3) INVALID_POSITION(4) POSITION_REACHED(5) PARAMETER_CHANGE_IN_EXECUTION(6) STEP_LIMIT(7) BLOCKED_BY_MODE(8) BLOCKED_BY_PROCESS(9) BLOCKED_BY_INTERLOCKING(10) BLOCKED_BY_SYNNCHROCHECK(11) COMMAND_ALREADY_IN_EXECUTION(12) BLOCKED_BY_HEALTH (13) ONE_OF_N_CONTROL(14) ABORTION_BY_CANCEL(15) TIME_LIMIT_OVER(16) ABORTION_BY_TRIP(17) OBJECT_NOT_SELECTED(18) OBJECT_ALREADY_SELECTED(19) NO_ACCESS_AUTHORITY(20) ENDED_WITH_OVERSHOOT(21) ABORTION_DUE_TO_DEVIATION(22) ABORTION_BY_COMMUNICATION_LOSS(23) BLOCKED_BY_COMAND(24) NONE(25) INCONSISTENT_PARAMETERS(26) LOCKED_BY_OTHER_CLIENT(27) HARDWARE_ERROR(28) UNKNOWN(29)
dwTimeout	DWORD	Time-out [ms] to the treatment by user logic

Table 72: CommandReceiver Function Block Input Parameters

Notes:

bExec: When FALSE, the command just stop being intercepted for the user application, but it keeps being treated normally by the server.

bDone: After the command interception, the user is going to be responsible for treat it. At the end of the treatment, this input must be enabled for a new command can be received. Case this input is not enabled, the block is going to wait the time defined in *dwTimeout*, to then become capable of intercept new commands.

eCommandResult: Treatment results of command intercepted by user. The result returned to the client that sent the command, which must be attributed together with the input *bDone*, is converted to the protocol's format from which was received the command. In Nexto Series it is only supported the interception of commands coming from protocol IEC 60870-5-104. In protocol interception, any return different from *SUCCESS* results in a negative Acknowledge.

ATTENTION

It is not recommended the simultaneous commands interception to one same variable by two or more *CommandReceiver* function blocks. Just one of the function blocks will intercept correctly the command, being able to suffer undesirable interference from the others function blocks if addressed to the same variable.

Parameter	Type	Description
bCommandAvailable	BOOL	Indicates that a command was intercepted and the data are available to be processed
sCommand	STRUCT	This structure stores received command data, which is composed by the following fields: eCommand sSelectParameters sOperateParameters The description of each field is in this section.
eStatus	ENUM (TYPE_RESULT)	Out of function action from obtained result, according to list: OK_SUCCESS(0) ERROR_FAILED(1)

Table 73: CommandReceiver Function Block Output Parameters

Note:

eStatus: Return of a register process of a communication point command interception. When the interception is registered with success *OK_SUCCESS* is returned, else *ERROR_FAILED* is. In case interceptor register failure, commands to the determined point are not intercepted by this function block. *TYPE_RESULT* is defined in *LibDataTypes* library.

Supported commands are described on table below:

Parameter	Type	Description
eCommand	ENUM	NO_COMMAND(0) SELECT(1) OPERATE(2)

Table 74: CommandReceiver Function Block Supported Commands

The parameters that build the *sSelectParameters*, *sOperateParameters* and *sCancelParameters* structures are described on the following table:

Parameter	Type	Description
sSelectConfig	STRUCT	Received selection command configuration. This structure parameters are described on Table 76
sValue	STRUCT	Received value in a select, when is received a selection command with value. This structure parameters are described on Table 79

Table 75: Parameters sSelectParameters

Parameter	Type	Description
bSelectWithValue	BOOL	When true indicates a selection command reception with value.

Table 76: Parameters sSelectConfig

Parameter	Type	Description
sOperateConfig	STRUCT	Received selection command configuration. This structure parameters are described on Table 78
sValue	STRUCT	Field of received operation command referred value. This structure parameters are described on Table 79

Table 77: Parameters sOperateParameters

Parameter	Type	Description
bDirectOperate	BOOL	When true indicates that an operation command without select was received.
bNoAcknowledgement	BOOL	When true indicates that a command, which doesn't require the receiving acknowledge, was received.
bTimedOperate	BOOL	When true indicates that an operation command activated by time was received.
liOperateTime	LINT	Programming of the instant in which it must be runned the command. This field is valid only when *bTimedOperate* is true.
bTest	BOOL	When true indicates that the received command was sent only for test, as so the command must not be runned.

Table 78: Parameters sOperateConfig

Parameter	Type	Description
eParamType	ENUM	Informs the type of the received command: NO_COMMAND(0) SINGLE_POINT_COMMAND(1) DOUBLE_POINT_COMMAND(2) INTEGER_STATUS_COMMAND(3) ENUMERATED_STATUS_COMMAND(4) ANALOGUE_VALUE_COMMAND(5)
sSinglePoint	STRUCT	When a command is received, in received command type function, defined by eParamType, the corresponding data structure is filled. This structures parameters are described on Tables 80 to 84
sDoublePoint	STRUCT	
sIntegerStatus	STRUCT	
sEnumeratedStatus	STRUCT	
sAnalogueValue	STRUCT	

Table 79: Parameters sValue

Parameter	Type	Description
bValue	BOOL	Point operation value.
sPulseConfig	STRUCT	The pulsed command configuration parameters are stored in this structure. This structure parameters are described on Table 85.

Table 80: Parameters sSinglePoint

Parameter	Type	Description
bValue	BOOL	Point operation value.
sPulseConfig	STRUCT	The pulsed command configuration parameters are stored in this structure. This structure parameters are described on Table 85.

Table 81: Parameters sDoublePoint

Parameter	Type	Description
diValue	DINT	Point operation value.

Table 82: Parameters sIntegerStatus

Parameter	Type	Description
dwValue	DWORD	Point operation value.

Table 83: Parameters sEnumeratedStatus

Parameter	Type	Description
eType	ENUM	Informs the data type of the received analog value. INTEGER (0) FLOAT (1)
diValue	DINT	Point operation value, integer format.
fValue	REAL	Point operation value, float format.

Table 84: Parameters sAnalogueValue

Parameter	Type	Description
bPulseCommand	BOOL	When true indicates that received command is pulsed.
dwOnDuration	DWORD	This is time, in milliseconds, that the output must remain on.
dwOffDuration	DWORD	This is time, in milliseconds, that the output must remain off.
dwPulseCount	DWORD	Number of times the command must be executed.

Table 85: Parameters sPulseConfig

To intercept commands to a specific point, first it is need to load in the *dwVariableAddr* parameter the variable address correspondent to the point wanted to intercept the commands and then execute a pulse in the *bExec* parameter. Once the command

was intercepted, the function block informs that a command was intercepted through *bCommandAvailable* parameter. The intercepted command information are then filled in the *sCommand* and *eStatus* output parameters, according to the received command type. This operation depends only of the received command type, don't matter the variable's data type to which is being intercepted the command. The interception is finished and then the function block can be released to intercept a new command when *bDone* parameter is *true*. Yet must be pointed the command processing result in *eCommandResult*.

5.5.5. MODBUS RTU Master

This protocol is available for the Nexto Series CPUs in its serial channels. By selecting this option at MasterTool IEC XE, the CPU becomes MODBUS communication master, allowing the access to other devices with the same protocol, when it is in the execution mode (*Run* Mode).

There are two configuration modes for this protocol. One makes use of Direct Representation (%Q), in which the variables are defined by its address. The other, called Symbolic Mapping has the variables defined by its name.

Regardless of the configuration mode, the steps to insert a protocol instance and configure the serial interface are the same. The procedure to insert a protocol instance is found in detail in the MasterTool IEC XE User Manual - MU299609 or in the section Inserting a Protocol Instance. The remaining configuration steps are described below for each mode.

- Add the MODBUS RTU Master Protocol instance to the serial channel COM 1 or COM 2 (or both, in case of two communication networks). To execute this procedure, see Inserting a Protocol Instance section.
- Configure the serial interface, choosing the transmission speed, the RTS/CTS signals behavior, the parity, the channel stop bits, among others configurations by a double click on the COM 1 or COM 2 serial channel. See Serial Interfaces Configuration section.

5.5.5.1. MODBUS Master Protocol Configuration by Symbolic Mapping

To configure this protocol using symbolic mapping, you must perform the following steps:

- Configure the general parameters of the MODBUS Master protocol, like: transmission delay times and minimum inter-frame as in Figure 53.
- Add and configure devices via the General Parameters tab, defining the slave address, communication time-out and number of communication retries as can be seen in Figure 54.
- Add and configure the MODBUS mappings on Mappings tab as Figure 55, specifying the variable name, data type, and the data initial address, the data size and range are filled automatically.
- Add and configure the MODBUS requests as presented in Figure 56, specifying the function, the scan time of the request, the starting address (read/write), the data size (read/write) and generate diagnostic variables and disabling the request via the buttons at the bottom of the window.

5.5.5.1.1. MODBUS Master Protocol General Parameters – Symbolic Mapping Configuration

The general parameters, found on the MODBUS protocol initial screen (figure below), are defined as:

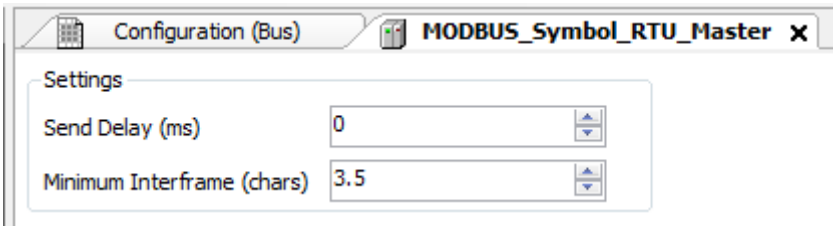


Figure 53: MODBUS RTU Master Configuration Screen

Configuration	Description	Default	Options
Send Delay (ms)	Delay for the answer transmission.	0	0 to 65535
Minimum Interframe (chars)	Minimum silence time between different frames.	3.5	3.5 to 100.0

Table 86: MODBUS RTU Master General Configurations

Notes:

Send Delay: The answer to a MODBUS protocol may cause problems in certain moments, as in the RS-485 interface or other half-duplex. Sometimes there is a delay between the slave answer time and the physical line silence (slave delay to put RTS in zero and put the RS-485 in high impedance state). To solve this problem, the master can wait the determined time in this field before sending the new request. Otherwise, the first bytes transmitted by the master could be lost.

Minimum Interframe: The MODBUS standard defines this time as 3.5 characters, but this parameter is configurable in order to attend the devices which do not follow the standard.

The MODBUS protocol diagnostics and commands configured, either by symbolic mapping or direct representation, are stored in *T_DIAG_MODBUS_RTU_MASTER_1* variables. For the direct representation mapping, they are also in 4 bytes and 8 words which are described in table below (where "n" is the configured value in the *%Q Start Address of Diagnostics Area* field):

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_MASTER_1.*	Size	Description
Diagnostics Bits:			
%QX(n).0	tDiag. bRunning	BIT	The master is running.
%QX(n).1	tDiag. bNotRunning	BIT	The master is not running (see bit: bInterruptedByCommand).
%QX(n).2	tDiag. bInterruptedByCommand	BIT	The bit bNotRunning was enabled as the master was interrupted by the user through command bits.
%QX(n).3	tDiag. bConfigFailure	BIT	Discontinued diagnosis.
%QX(n).4	tDiag. bRXFailure	BIT	Discontinued diagnosis.
%QX(n).5	tDiag. bTXFailure	BIT	Discontinued diagnosis.
%QX(n).6	tDiag. bModuleFailure	BIT	Indicates if there is failure in the module or the module is not present.
%QX(n).7	tDiag. bDiag_7_reserved	BIT	Reserved
Error Codes:			
			0: there are no errors 1: invalid serial port 2: invalid serial port mode 3: invalid baud rate 4: invalid data bits 5: invalid parity 6: invalid stop bits 7: invalid modem signal parameter

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_MASTER_1.*	Size	Description
%QB(n+1)	eErrorCode	SERIAL_STATUS (BYTE)	8: invalid UART RX Threshold parameter 9: invalid time-out parameter 10: busy serial port 11: UART hardware error 12: remote hardware error 20: invalid transmission buffer size 21: invalid signal modem method 22: CTS time-out = true 23: CTS time-out = false 24: transmission time-out error 30: invalid reception buffer size 31: reception time-out error 32: flow control configured differently from manual 33: invalid flow control for the configured serial port 34: data reception not allowed in normal mode 35: data reception not allowed in extended mode 36: DCD interruption not allowed 37: CTS interruption not allowed 38: DSR interruption not allowed 39: serial port not configured 50: internal error in the serial port
Command bits, automatically initialized:			
%QX(n+2).0	tCommand. bStop	BIT	Stop master.
%QX(n+2).1	tCommand. bRestart	BIT	Restart master.
%QX(n+2).2	tCommand. bResetCounter	BIT	Restart diagnostics statistics (counters).
%QX(n+2).3	tCommand. bDiag_19_reserved	BIT	Reserved
%QX(n+2).4	tCommand. bDiag_20_reserved	BIT	Reserved
%QX(n+2).5	tCommand. bDiag_21_reserved	BIT	Reserved
%QX(n+2).6	tCommand. bDiag_22_reserved	BIT	Reserved
%QX(n+2).7	tCommand. bDiag_23_reserved	BIT	Reserved
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved
Communication Statistics:			
%QW(n+4)	tStat. wTXRequests	WORD	Counter of request transmitted by the master (0 to 65535).

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_MASTER_1.*	Size	Description
%QW(n+6)	tStat. wRXNormalResponses	WORD	Counter of normal responses received by the master (0 to 65535).
%QW(n+8)	tStat. wRXExceptionResponses	WORD	Counter of responses with exception codes received by the master (0 to 65535).
%QW(n+10)	tStat. wRXIllegalResponses	WORD	Counter of illegal responses received by master – invalid syntax, not enough received bytes, invalid CRC – (0 to 65535).
%QW(n+12)	tStat. wRXOverrunErrors	WORD	Counter of overrun errors during reception - UART FIFO or RX line – (0 to 65535).
%QW(n+14)	tStat. wRXIncompleteFrames	WORD	Counter of answers with construction errors, parity or failure during reception (0 to 65535).
%QW(n+16)	tStat. wCTSTimeOutErrors	WORD	Counter of CTS time-out error, using RTS/CTS handshake, during transmission (0 to 65535).
%QW(n+18)	tStat. wDiag_18_Reserved	WORD	Reserved

Table 87: MODBUS RTU Master Diagnostics

Note:
Counters: All MODBUS RTU Master diagnostics counters return to zero when the limit value 65535 is exceeded.

5.5.5.1.2. *Devices Configuration – Symbolic Mapping configuration*

The devices configuration, shown on figure below, follows the following parameters:

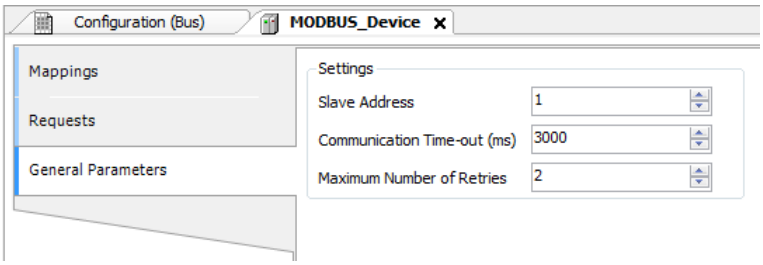


Figure 54: Device General Parameters Settings

Configuration	Description	Default	Options
Slave Address	MODBUS slave address	1	0 to 255
Communication Time-out (ms)	Defines the application level time-out	3000	10 to 65535
Maximum Number of Retries	Defines the numbers of retries before reporting a communication error	2	0 to 9

Table 88: Device Configurations

Notes:

Slave Address: According to the MODBUS standard, the valid slave addresses are from 0 to 247, where the addresses from 248 to 255 are reserved. When the master sends a writing command with the address configured as zero, it is making broadcast requests in the network.

Communication Time-out: The communication time-out is the time that the master waits for a response from the slave to the request. For a MODBUS RTU master device it must be taken into account at least the following system variables: the time it takes the slave to transmit the frame (according to the baud rate), the time the slave takes to process the request and the response sending delay if configured in the slave. It is recommended that the time-out is equal to or greater than the time to transmit the frame plus the delay of sending the response and twice the processing time of the request. For more information, see Communication Performance section.

Maximum number of retries: Sets the number of retries before reporting a communication error. For example, if the slave does not respond to a request and the master is set to send three retries, the error counter number is incremented by one unit when the execution of these three retries. After the increase of the communication error trying to process restarts and if the number of retries is reached again, new error will increment the counter.

5.5.5.1.3. Mappings Configuration – Symbolic Mapping Settings

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:

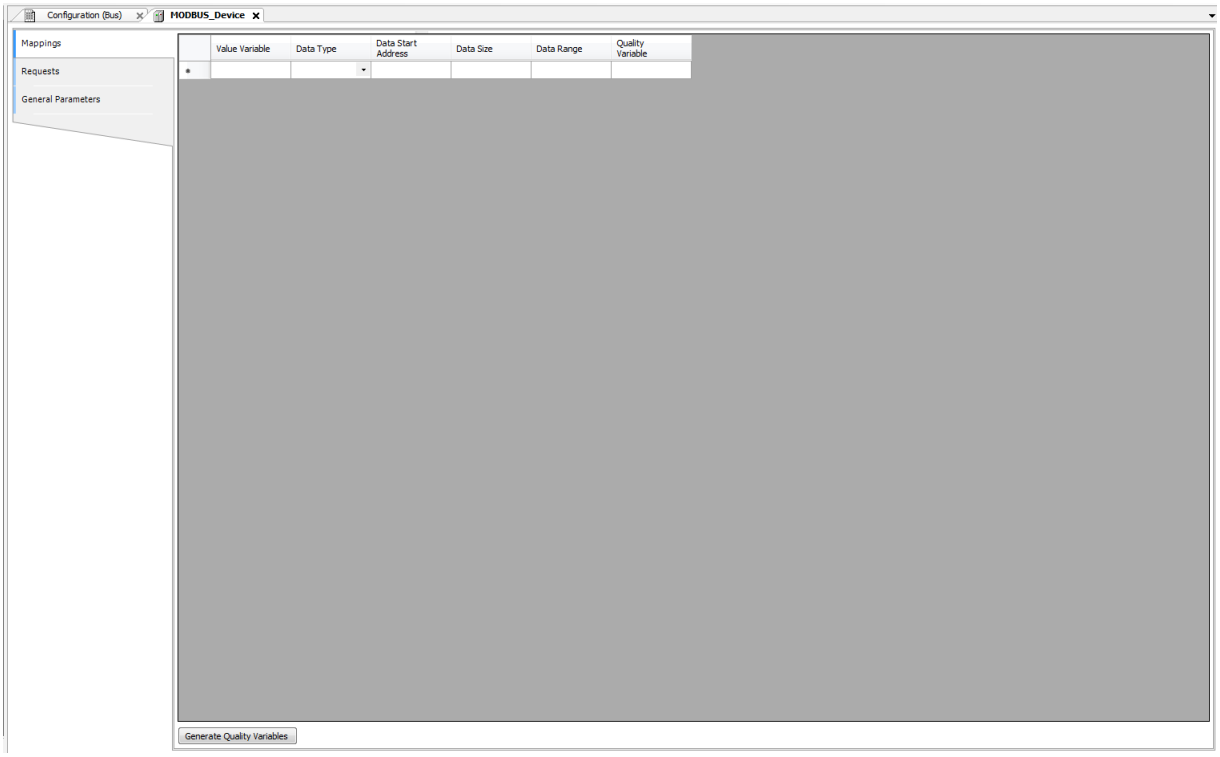


Figure 55: MODBUS Data Mappings Screen

Configuration	Description	Default	Options
Value Variable	Symbolic variable name	-	Name of a variable declared in a program or GVL
Data Type	MODBUS data type	-	Coil - Write (1 bit) Coil - Read (1 bit) Holding Register - Write (16 bits) Holding Register - Read (16 bits) Holding Register – Mask AND (16 bits) Holding Register – Mask OR (16 bits) Input Register (16 bits) Input Status (1 bit)
Data Start Address	Initial address of the MODBUS data	-	1 to 65536
Data Size	Size of the MODBUS data	-	1 to 65536
Data Range	The address range of configured data	-	-

Table 89: MODBUS Mappings Settings

Notes:

Value Variable: this field is used to specify a symbolic variable in MODBUS relation.

Data type: this field is used to specify the data type used in the MODBUS relation.

Data Type	Size [bits]	Description
Coil - Write	1	Writing digital output.
Coil - Read	1	Reading digital output.
Holding Register - Write	16	Writing analog output.
Holding Register - Read	16	Reading analog output.
Holding Register - Mask AND	16	Analog output which can be read or written with AND mask.
Holding Register - Mask OR	16	Analog output which can be read or written with OR mask.
Input Register	16	Analog input which can be only read.
Input Status	1	Digital input which can be only read.

Table 90: Data Types Supported in MODBUS

Data Start Address: Data initial address of a MODBUS mapping.

Data Size: The size value specifies the maximum amount of data that a MODBUS interface can access, from the initial address. Thus, to read a continuous address range, it is necessary that all addresses are declared in a single interface. This field varies with the MODBUS data type configured.

Data Range: This field shows to the user the memory address range used by the MODBUS interface.

5.5.5.1.4. Requests Configuration – Symbolic Mapping Settings

The configuration of the MODBUS requests, viewed in figure below, follow the parameters described in table below:

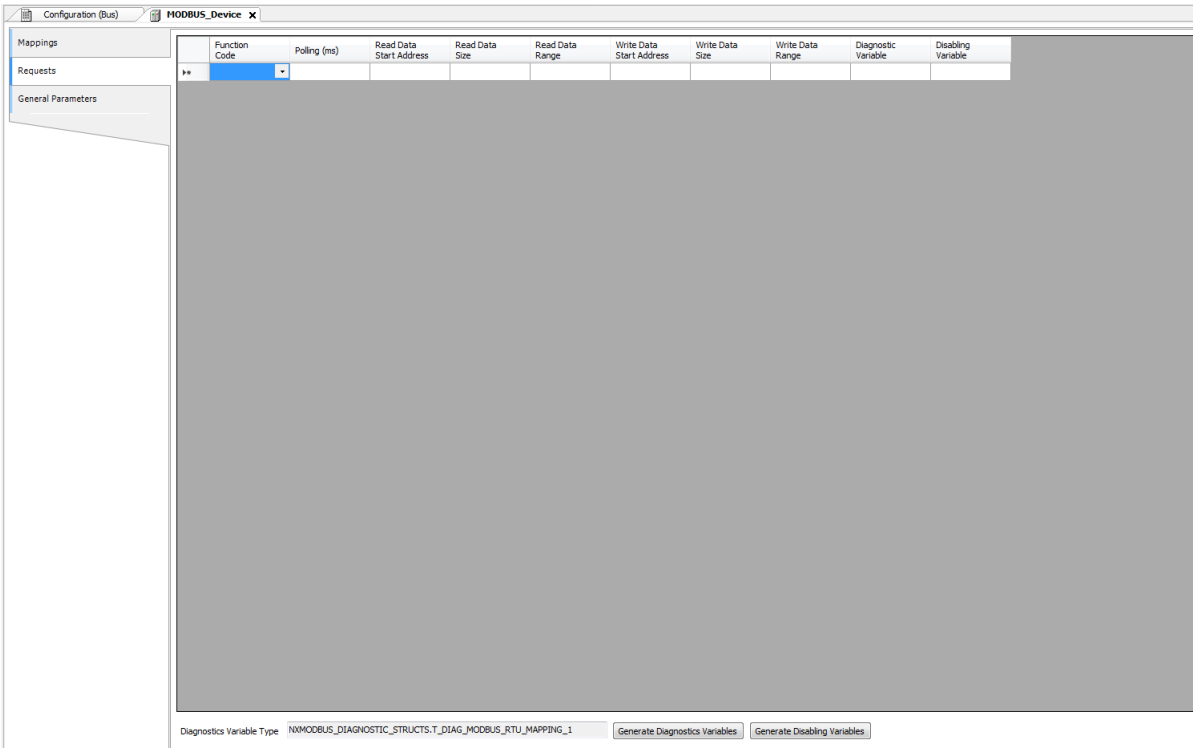


Figure 56: Data Requests Screen MODBUS Master

Configuration	Description	Default Value	Options
Function Code	MODBUS function type	-	01 – Read Coils 02 – Read Input Status 03 – Read Holding Registers 04 – Read Input Registers 05 – Write Single Coil 06 – Write Single Register 15 – Write Multiple Coils 16 – Write Multiple Registers 22 – Mask Write Register 23 – Read/Write Multiple Registers
Polling (ms)	Communication period (ms)	100	0 to 3600000
Read Data Start Address	Initial address of the MODBUS read data	-	1 to 65536
Read Data Size	Size of MODBUS Read data	-	Depends on the function used
Read Data Range	MODBUS Read data address range	-	0 to 2147483646
Write Data Start Address	Initial address of the MODBUS write data	-	1 to 65536
Write Data Size	Size of MODBUS Write data	-	Depends on the function used
Write Data Range	MODBUS Write data address range	-	0 to 2147483647

Configuration	Description	Default Value	Options
Diagnostic Variable	Diagnostic variable name	-	Name of a variable declared in a program or GVL
Disabling Variable	Variable used to disable MODBUS relation	-	Field for symbolic variable used to disable, individually, MODBUS requests configured. This variable must be of type BOOL. The variable can be simple or array element and can be in structures.

Table 91: MODBUS Relations Configuration

Notes:

Setting: the number of factory default settings and the values for the column Options may vary according to the data type and MODBUS function (FC).

Function Code: MODBUS (FC) functions available are the following:

Code		Description
DEC	**HEX**	
1	0x01	Read Coils (FC 01)
2	0x02	Read Input Status (FC 02)
3	0x03	Read Holding Registers (FC 03)
4	0x04	Read Input Registers (FC 04)
5	0x05	Write Single Coil (FC 05)
6	0x06	Write Single Holding Register (FC 06)
15	0x0F	Write Multiple Coils (FC 15)
16	0x10	Write Multiple Holding Registers (FC 16)
22	0x16	Mask Write Holding Register (FC 22)
23	0x17	Read/Write Multiple Holding Registers (FC 23)

Table 92: MODBUS Functions Supported by Nexto CPUs

Polling: this parameter indicates how often the communication set for this request must be performed. By the end of a communication will be awaited a time equal to the value configured in the field polling and after that, a new communication will be executed.

Read Data Start Address: field for the initial address of the MODBUS read data.

Read Data Size: the minimum value for the read data size is 1 and the maximum value depends on the MODBUS function (FC) used as below:

- Read Coils (FC 01): 2000
- Read Input Status (FC 02): 2000
- Read Holding Registers (FC 03): 125
- Read Input Registers (FC 04): 125
- Read/Write Multiple Registers (FC 23): 121

Read Data Range: this field shows the MODBUS read data range configured for each request. The initial address, along with the read data size will result in the range of read data for each request.

Write Data Start Address: field for the initial address of the MODBUS write data.

Write Data Size: the minimum value for the write data size is 1 and the maximum value depends on the MODBUS function (FC) used as below:

- Write Single Coil (FC 05): 1

- Write Single Register (FC 06): 1
- Write Multiple Coils (FC 15): 1968
- Write Multiple Registers (FC 16): 123
- Mask Write Register (FC 22): 1
- Read/Write Multiple Registers (FC 23): 121

Write Data Range: this field shows the MODBUS write data range configured for each request. The initial address, along with the read data size will result in the range of write data for each request.

Diagnostic Variable: The MODBUS request diagnostics configured by symbolic mapping or by direct representation, are stored in variables of type *T_DIAG_MODBUS_RTU_MAPPING_1* for Master devices and *T_DIAG_MODBUS_ETH_CLIENT_1* for Client devices and the mapping by direct representation are in 4-byte and 2-word, which are described in Table 93 ("n" is the value configured in the *%Q Start Address of Diagnostics Area* field).

Direct Representation Variable	Diagnostic variable of type T_DIAG_MODBUS _RTU_MAPPING_1.*	Size	Description
Communication status bits:			
%QX(n).0	byStatus. bCommIdle	BIT	Communication idle (waiting to be executed).
%QX(n).1	byStatus. bCommExecuting	BIT	Active communication.
%QX(n).2	byStatus. bCommPostponed	BIT	Communication delayed, because the maximum number of concurrent requests was reached. Deferred communications will be carried out in the same sequence in which they were ordered to avoid indeterminacy. The time spent in this State is not counted for the purposes of time-out. The bCommIdle and bCommExecuting bits are false when the bCommPostponed bit is true.
%QX(n).3	byStatus. bCommDisabled	BIT	Communication disabled. The bCommIdle bit is restarted in this condition.
%QX(n).4	byStatus. bCommOk	BIT	Communication terminated previously was held successfully.
%QX(n).5	byStatus. bCommError	BIT	Communication terminated previously had an error. Check error code.
%QX(n).6	byStatus. bCommAborted	BIT	Not used in MODBUS RTU Master.
%QX(n).7	byStatus. bDiag_7_reserved	BIT	Reserved
Last error code (enabled when bCommError = true):			
%QB(n+1)	eLastErrorCode	MASTER_ERROR _CODE (BYTE)	Informs the possible cause of the last error in the MODBUS mapping. Consult Table 116 for further details.
Last exception code received by master:			
%QB(n+2)	eLastExceptionCode	MODBUS _EXCEPTION (BYTE)	NO_EXCEPTION (0) FUNCTION_NOT_SUPPORTED (1) MAPPING_NOT_FOUND (2) ILLEGAL_VALUE (3) ACCESS_DENIED (128)* MAPPING_DISABLED (129)* IGNORE_FRAME (255)*

Direct Representation Variable	Diagnostic variable of type T_DIAG_MODBUS _RTU_MAPPING_1.*	Size	Description
Communication statistics:			
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
%QW(n+4)	wCommCounter	WORD	Finished communications counter (with or without errors). The user can test when communication has finished testing the variation of this counter. When the value 65535 is reached, the counter returns to zero.
%QW(n+6)	wCommErrorCounter	WORD	Finished communications counter (with errors). When the value 65535 is reached, the counter returns to zero.

Table 93: MODBUS Relations Diagnostics

Notes:

Exception Codes: The exception codes presented in this field are values returned by the slave. The definitions of the exception codes 128, 129 and 255 presented in the table are valid only when using Altus slaves. Slaves from other manufacturers might use other definitions for each code.

Disabling Variable: variable of Boolean type used to disable, individually, MODBUS requests configured on request tab via button at the bottom of the window. The request is disabled when the variable, corresponding to the request, is equal to 1, otherwise the request is enabled.

Last Error Code: The codes for the possible situations that cause an error in the MODBUS communication can be consulted below:

Code	Enumerable	Description
1	ERR_EXCEPTION	Reply is in an exception code (see eLastExceptionCode = Exception Code).
2	ERR_CRC	Reply with invalid CRC.
3	ERR_ADDRESS	MODBUS address not found. The address that replied the request was different than expected.
4	ERR_FUNCTION	Invalid function code. The reply's function code was different than expected.
5	ERR_FRAME_DATA_COUNT	The amount of data in the reply was different than expected.
7	ERR_NOT_ECHO	The reply is not an echo of the request (FC 05 and 06).
8	ERR_REFERENCE_NUMBER	Invalid reference number (FC 15 and 16).
9	ERR_INVALID_FRAME_SIZE	Reply shorter than expected.
20	ERR_CONNECTION	Error while establishing connection.
21	ERR_SEND	Error during transmission stage.
22	ERR_RECEIVE	Error during reception stage.
40	ERR_CONNECTION_TIMEOUT	Application level time-out during connection.
41	ERR_SEND_TIMEOUT	Application level time-out during transmission.
42	ERR_RECEIVE_TIMEOUT	Application level time-out while waiting for reply.
43	ERR_CTS_OFF_TIMEOUT	Time-out while waiting CTS = false in transmission.
44	ERR_CTS_ON_TIMEOUT	Time-out while waiting CTS = true in transmission.
128	NO_ERROR	No error since startup.

Table 94: MODBUS Relations Error Codes

ATTENTION

Differently from other application tasks, when a depuration mark in the MainTask is reached, the task of a Master MODBUS RTU instance and any other MODBUS task will stop running at the moment that it tries to perform a writing in a memory area. It occurs in order to keep the consistency of the memory areas data while a MainTask is not running.

5.5.5.2. MODBUS Master Protocol Configuration for Direct Representation (%Q)

To configure this protocol using direct representation (%Q), the following steps must be performed:

- Configure the general parameters of the MODBUS protocol, such as: communication times and direct representation variables (%Q) to receive diagnostics.
- Add and configure devices by setting address, direct representation variables (%Q) to disable the relations, communication time-outs, etc.
- Add and configure MODBUS relations, specifying the data type and MODBUS function, time-outs, direct representation variables (%Q) to receive diagnostics of the relation and other to receive/write the data, amount of data to be transmitted and relation polling.

The descriptions of each configuration are listed below in this section.

5.5.5.2.1. General Parameters of MODBUS Master Protocol - setting by Direct Representation (%Q)

The General parameters, found on the home screen of MODBUS protocol configuration (figure below), are defined as:

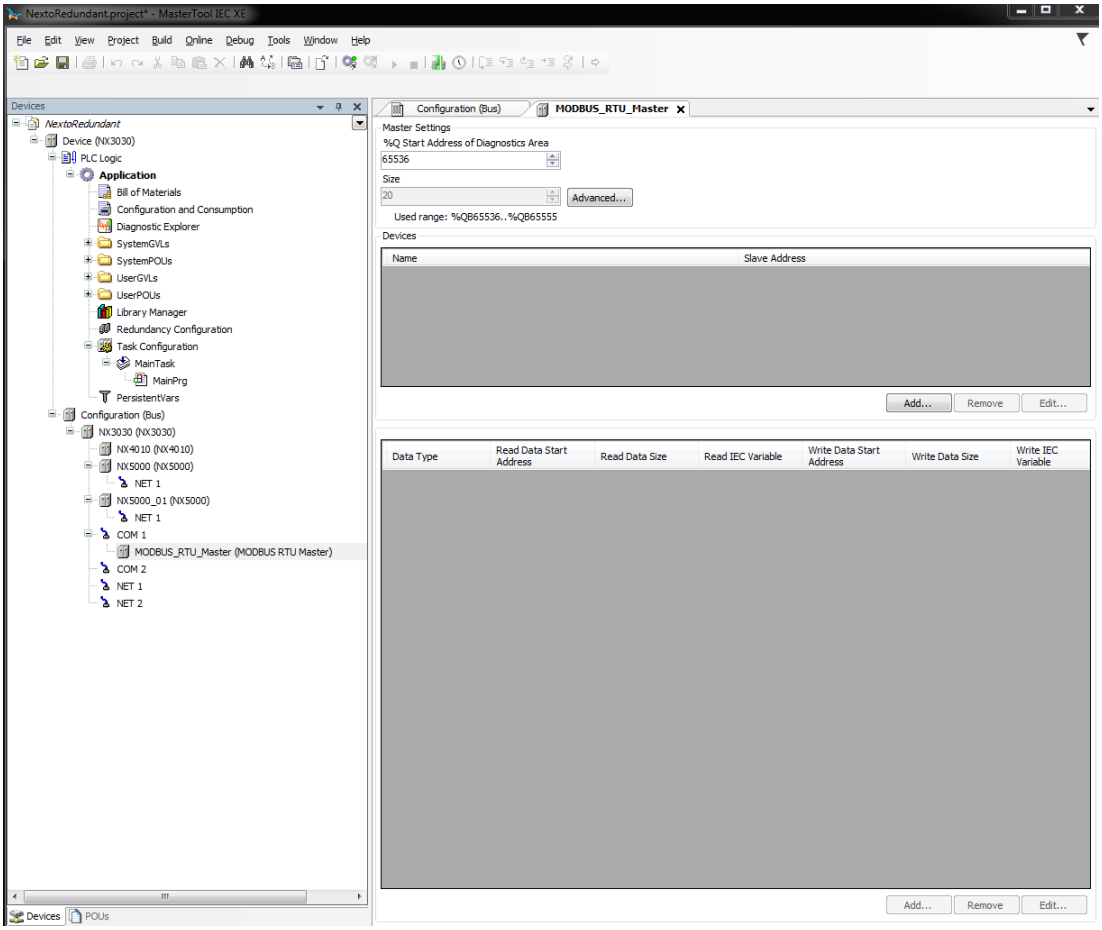


Figure 57: MODBUS RTU Master Setup Screen

Direct representation variables (%Q) for the protocol diagnostic:

Configuration	Description	Default Value	Options
%Q Start Address of Diagnostics Area	Initial address of the diagnostic variables	-	0 to 2147483628
Size	Size of diagnostics area	20	Disabled for editing

Table 95: MODBUS RTU Master Configuration

Notes:

Initial Address of Diagnostics in %Q: this field is limited by the size of outputs variables (%Q) addressable memory of each CPU, which can be found in section Memory.

Default Value: the factory default value cannot be set to the *%Q Start Address of Diagnostics Area* field, because the creation of a Protocol instance may be held at any time on application development. The MasterTool IEC XE software itself allocate a value, from the range of output variables of direct representation (%Q), not used yet.

The diagnostics and MODBUS protocol commands are described in Table 87.

The communication times of the MODBUS Master protocol, found on the button *Advanced...* in the configuration screen are divided into *Send Delay* and *Minimum Interframe*, further details are described in section MODBUS Master Protocol General Parameters – Symbolic Mapping Configuration.

5.5.5.2.2. *Devices Configuration – Configuration for Direct Representation (%Q)*

The configuration of the devices, viewed in figure below, comprises the following parameters:

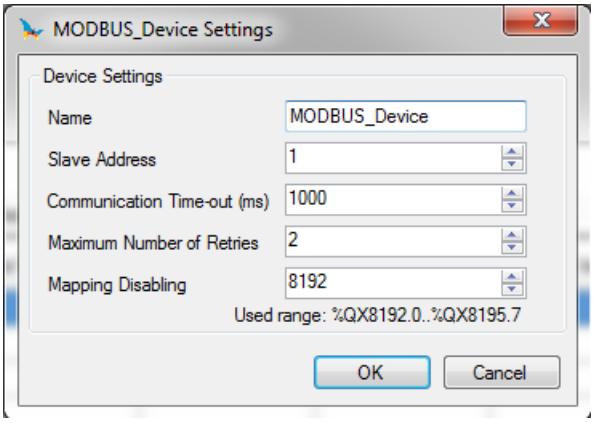


Figure 58: Device Configuration

Configuration	Description	Default Value	Options
Name	Name of the instance	MODBUS_Device	Identifier, according to IEC 61131-3
Slave Address	The MODBUS slave address	1	0 to 255
Communication Time-out (ms)	Sets the time-out of the application level	1000	10 to 65535
Maximum Number of Retries	Sets the number of retries before reporting a communication error	2	0 to 9

Configuration	Description	Default Value	Options
Mapping Disabling	Initial address used to disable MODBUS relations	-	0 to 2147483644

Table 96: Device Configuration - MODBUS Master

Notes:

Instance Name: this field is the identifier of the device, which is checked according to IEC 61131-3, i.e. does not allow spaces, special characters and start with numeral character. It's limited in 24 characters.

Mapping Disabling: composed of 32 bits, used to disable, individually, the 32 MODBUS relations configured in *Device Mappings* space. The relation is disabled when the bit, corresponding to relation, is equal to 1, otherwise, the mapping is enabled. This field is limited by the size of outputs variables (%Q) addressable memory of each CPU, which can be found in section Memory.

Default Value: the factory default value cannot be set to the *Mapping Disabling* field, because the creation of a Protocol instance may be held at any time on application development. The MasterTool IEC XE software itself allocate a value, from the range of output variables of direct representation (%Q), not used yet.

For further details on the *Slave Address*, *Communication Time-out* and *Maximum Number of Retries* parameters see notes in section Devices Configuration – Symbolic Mapping configuration.

5.5.5.2.3. *Mappings Configuration – Configuration for Direct Representation (%Q)*

The MODBUS relations settings, viewed in the figures below, follow the parameters described in table below:

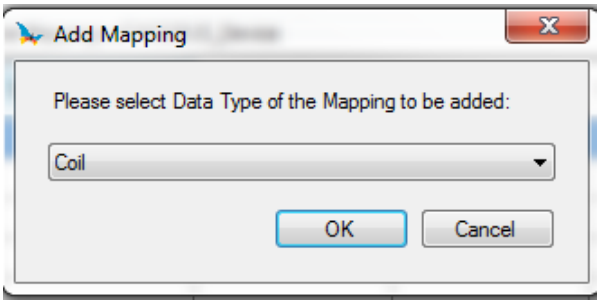


Figure 59: MODBUS Data Type

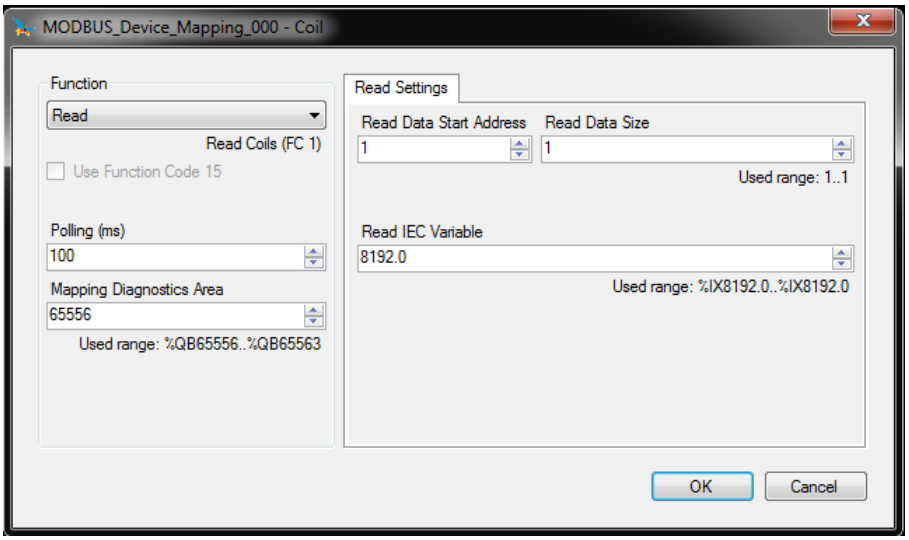


Figure 60: MODBUS Function

In table below, the number of factory default settings and the values for the column Options, may vary according to the data type and MODBUS function (FC).

Configuration	Description	Default Value	Options
Function	MODBUS function type	Read	Read Write Read/Write Mask Write
Polling (ms)	Communication period (ms)	100	0 to 3600000
Mapping Diagnostics Area	Initial address of the MODBUS relation diagnostics (%Q)	-	0 to 2147483640
Read Data Start Address	Initial address of the MODBUS read data	1	1 to 65536
Read Data Size	Number of MODBUS read data	-	Depends on the function used
Read IEC Variable	Initial address of the read variables (%I)	-	0 to 2147483646
Write Data Start Address	Initial address of the MODBUS write data	1	1 to 65536
Write Data Size	Number of MODBUS write data	-	Depends on the function used
Write IEC Variable	Initial address of the write variables (%Q)	-	0 to 2147483647
Mask Write IEC Variables	Initial address of the variables for the write mask (%Q)	-	0 to 2147483644

Table 97: Device Mapping

Notes:

Function: the available data types are detailed in the Table 116 and MODBUS functions (FC) are available in the Table 114.

Polling: this parameter indicates how often the communication set for this relation must be executed. At the end of communication will be awaited a time equal to the configured polling and after, will be performed a new communication as soon as possible.

Mapping Diagnostics Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in the section Memory. The configured MODBUS relations diagnostics are described in Table 93.

Read/Write Data Size: details of the data size supported by each function are described in the notes of the section Requests Configuration – Symbolic Mapping Settings.

ATTENTION

When accessing the communication data memory is between devices with different endianism (Little-Endian and Big-Endian), inversion of the read/write data may occur. In this case, the user must adjust the data in the application.

Read IEC Variable: if the MODBUS data type is *Coil* or *Input Status* (bit), the initial address of the IEC reading variables will have the format *%IX10.1*, for example. However, if the MODBUS data type is *Holding Register* or *Input Register* (16 bits), the initial address of the IEC reading variables will be *%IW*. This field is limited by the size of input variables addressable memory *(%I)* at CPU, which can be found in the section Memory.

Write IEC Variable: if the MODBUS data type is *Coil*, the initial address of the IEC writing variables will have the format *%QX10.1*, for example. However, if the MODBUS data type is *Holding Register* (16 bits), the initial address of the IEC writing variables will be *%QW*. This field is limited by the size of output variables addressable memory *(%Q)* at CPU, which can be found in the section Memory.

Write Mask: the function *Mask Write* (FC 22), employs a logic between the value already written and the two words that are configured in this field using *%QW(0)* for the AND mask and *%QW(2)* for the OR mask; allowing the user to handle the word. This field is limited by the size of output variables addressable memory *(%Q)* of each CPU, which can be found in the section Memory.

Default Value: the factory default value cannot be set for the *Mapping Diagnostics Area*, *Read IEC Variable*, *Write IEC Variable* and *Mask Write IEC Variables* fields, since the creation of a relation can be performed at any time on application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables *(%Q)*, still unused. Factory default cannot be set to the *Read/Write Data Size* fields, as they will vary according to the MODBUS data type selected.

ATTENTION

Unlike other tasks of an application, when a mark is reached at MainTask debugging, the MODBUS RTU Master instance task or any other MODBUS task will stop being executed at the moment it tries to write in the memory area. This occurs in order to maintain data consistency of memory areas while MainTask is not running.

5.5.6. MODBUS RTU Slave

This protocol is available for the Nexto Series on its serial channels. At selecting this option in MasterTool IEC XE, the CPU becomes a MODBUS communication slave, allowing the connection with MODBUS RTU master devices.

There are two ways to configure this protocol. The first one makes use of direct representation (%Q), in which the variables are defined by your address. The second one, through symbolic mapping, where the variables are defined by your name.

Independent of the configuration mode, the steps to insert an instance of the protocol and configure the serial interface are equal. The procedure to insert an instance of the protocol is found in detail in the MasterTool IEC XE User Manual - MU299609. The remaining configuration steps are described below for each mode:

- Add the MODBUS RTU Slave Protocol instance to the serial channel COM 1 or COM 2 (or both, in cases of two communication networks). To execute this procedure see Inserting a Protocol Instance section.
- Configure the serial interface, choosing the communication speed, the RTS/CTS signals behavior, the parity, the stop bits channel, among others. See Serial Interfaces Configuration section.

5.5.6.1. MODBUS Slave Protocol Configuration via Symbolic Mapping

To configure this protocol using symbolic mapping, you must perform the following steps:

- Configure the MODBUS slave protocol general parameters, as: slave address and communication times (available at the Slave advanced configurations button).
- Add and configure MODBUS relations, specifying the variable name, MODBUS data type and data initial address. Automatically, the data size and range will be filled, in accordance to the variable type declared.

5.5.6.1.1. MODBUS Slave Protocol General Parameters – Configuration via Symbolic Mapping

The general parameters, found on the MODBUS protocol initial screen (figure below), are defined as.

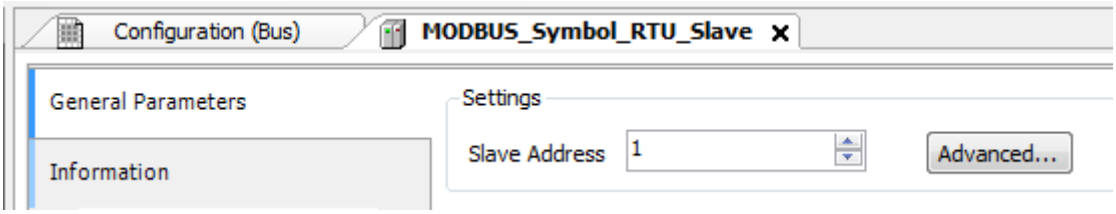


Figure 61: MODBUS RTU Slave Configuration Screen

Configuration	Description	Default	Options
Slave Address	MODBUS slave address	1	1 to 255

Table 98: Slave Configurations

The MODBUS slave protocol communication times, found in the *Advanced...* button on the configuration screen, are divided in: *Task Cycle*, *Send Delay* and *Minimum Interframe* as shown in figure below and in table below.

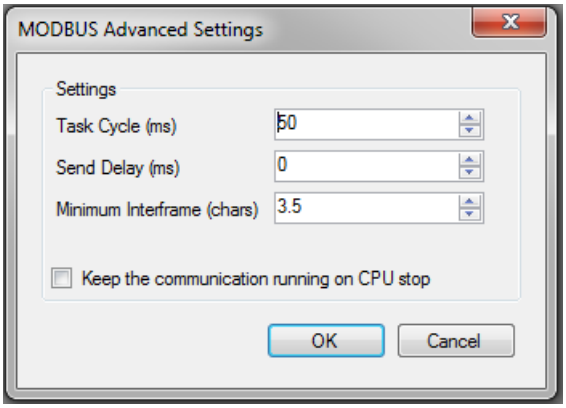


Figure 62: Modbus Slave Advanced Configurations

Configuration	Description	Default	Options
Task Cycle (ms)	Time for the instance execution within the cycle, without considering its own execution time	50	20 to 100
Send Delay (ms)	Delay for the transmission response	0	0 to 65535
Minimum Interframe (chars)	Minimum silence time between different frames	3.5	3.5 to 100.0
Keep the communication running on CPU stop	Enable the MODBUS Symbol Slave to run while the CPU is in STOP or standing in a breakpoint	Unchecked	Checked or unchecked

Table 99: Modbus Slave Advanced Configurations

Notes:

Task Cycle: the user will have to be careful when changing this parameter as it interferes directly in the answer time, data volume for scan and mainly in the CPU resources balance between communications and other tasks.

Send Delay: the answer to a MODBUS protocol may cause problems in certain moments, as in the RS-485 interface or other half-duplex. Sometimes there is a delay between the time of the request from the master and the silence on the physical line (slave delay to put RTS in zero and put the RS-485 in high impedance state). To solve this problem, the master can wait the determined time in this field before sending the new request. On the opposite case, the first bytes transmitted by the master could be lost.

Minimum Interframe: the MODBUS standard defines this time as 3.5 characters, but this parameter is configurable in order to attend the devices which don't follow the standard.

The MODBUS Slave protocol diagnostics and commands configured, either by symbolic mapping or direct representation, are stored in *T_DIAG_MODBUS_RTU_SLAVE_1* variables. For the direct representation mapping, they are also in 4 bytes and 8 words which are described in table below (where "n" is the configured value in the *%Q Start Address of Diagnostics Area*):

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_SLAVE_1.*	Size	Description
Diagnostic Bits:			
%QX(n).0	tDiag. bRunning	BIT	The slave is in execution mode.
%QX(n).1	tDiag. bNotRunning	BIT	The slave is not in execution (see bit: bInterruptedByCommand).
%QX(n).2	tDiag. bInterruptedByCommand	BIT	The bit bNotRunning was enabled as the slave was interrupted by the user through command bits.
%QX(n).3	tDiag. bConfigFailure	BIT	Discontinued diagnosis.
%QX(n).4	tDiag. bRXFailure	BIT	Discontinued diagnosis.
%QX(n).5	tDiag. bTXFailure	BIT	Discontinued diagnosis.
%QX(n).6	tDiag. bModuleFailure	BIT	Discontinued diagnosis.
%QX(n).7	tDiag. bDiag_7_reserved	BIT	Reserved.
Error codes:			
%QB(n+1)	eErrorCode	SERIAL_STATUS (BYTE)	0: there are no errors 1: invalid serial port 2: invalid serial port mode 3: invalid baud rate 4: invalid data bits 5: invalid parity 6: invalid stop bits 7: invalid modem signal parameter 8: invalid UART RX Threshold parameter 9: invalid time-out parameter 10: busy serial port 11: UART hardware error 12: remote hardware error 20: invalid transmission buffer size 21: invalid signal modem method 22: CTS time-out = true 23: CTS time-out = false 24: transmission time-out error 30: invalid reception buffer size 31: reception time-out error 32: flow control configured differently from manual 33: invalid flow control for the configured serial port 34: data reception not allowed in normal mode 35: data reception not allowed in extended mode 36: DCD interruption not allowed

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_SLAVE_1.*	Size	Description
			37: CTS interruption not allowed 38: DSR interruption not allowed 39: serial port not configured 50: internal error in the serial port
Command bits, automatically initialized:			
%QX(n+2).0	tCommand. bStop	BIT	Stop slave.
%QX(n+2).1	tCommand. bRestart	BIT	Restart slave.
%QX(n+2).2	tCommand. bResetCounter	BIT	Restart diagnostics statistics (counters).
%QX(n+2).3	tCommand. bDiag_19_reserved	BIT	Reserved.
%QX(n+2).4	tCommand. bDiag_20_reserved	BIT	Reserved.
%QX(n+2).5	tCommand. bDiag_21_reserved	BIT	Reserved.
%QX(n+2).6	tCommand. bDiag_22_reserved	BIT	Reserved.
%QX(n+2).7	tCommand. bDiag_23_reserved	BIT	Reserved.
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
Communication Statistics:			
%QW(n+4)	tStat. wRXRequests	WORD	Counter of normal requests received by the slave and answered normally. In case of a broadcast command, this counter is incremented, but it is not transmitted (0 to 65535).
%QW(n+6)	tStat. wTXExceptionResponses	WORD	Counter of normal requests received by the slave and answered with exception code. In case of a broadcast command, this counter is incremented, but it isn't transmitted (0 to 65535). Exception codes: 1: the function code (FC) is legal, but not supported. 2: relation not found in these MODBUS data. 3: illegal value for this field. 128: the master/client hasn't right for writing or reading. 129: the MODBUS relation is disabled.
%QW(n+8)	tStat. wRXFrames	WORD	Counter of frames received by the slave. It's considered a frame something which is processed and it is followed by a Minimum Interframe Silence, in other words, an illegal message is also computed (0 to 65535).

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _RTU_SLAVE_1.*	Size	Description
%QW(n+10)	tStat. wRXIllegalRequests	WORD	Illegal request counter. These are frames which start with address 0 (broadcast) or with the MODBUS slave address, but aren't legal requests – invalid syntax, smaller frames, invalid CRC – (0 to 65535).
%QW(n+12)	tStat. wRXOverrunErrors	WORD	Counter of frames with overrun errors during reception – UART FIFO or RX line – (0 to 65535).
%QW(n+14)	tStat. wRXIncompleteFrames	WORD	Counter of frames with construction errors, parity or failure during reception (0 to 65535).
%QW(n+16)	tStat. wCTSTimeOutErrors	WORD	Counter of CTS time-out error, using the RTS/CTS handshake, during the transmission (0 to 65535).
%QW(n+18)	tStat. wDiag_18_Reserved	WORD	Reserved.

Table 100: MODBUS RTU Slave Diagnostic

Note:
Counters: all MODBUS RTU Slave diagnostics counters return to zero when the limit value 65535 is exceeded.

5.5.6.1.2. Configuration of the Relations – Symbolic Mapping Setting

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:

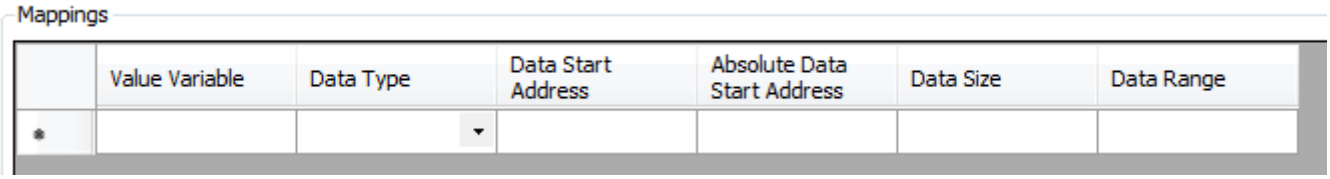


Figure 63: MODBUS Data Mappings Screen

Configuration	Description	Default	Options
Value Variable	Symbolic variable name	-	Name of a variable declared in a program or GVL
Data Type	MODBUS data type	-	Coil Input Status Holding Register Input Register
Data Start Address	MODBUS data initial address	-	1 to 65536

Configuration	Description	Default	Options
Absolute Data Start Address	Absolute initial address of MODBUS data according to its type	-	-
Data Size	MODBUS data size	-	1 to 65536
Data Range	Data address range configured	-	-

Table 101: MODBUS Mappings Configurations

Notes:

Value Variable: this field is used to specify a symbolic variable in MODBUS relation.

Data Type: this field is used to specify the data type used in the MODBUS relation.

Data Type	Size [bits]	Description
Coil	1	Digital output that can be read or written.
Input Status	1	Digital input (read only).
Holding Register	16	Analog output that can be read or written.
Input Register	16	Analog input (read only).

Table 102: MODBUS data types supported by Nexto CPUs

Data Start Address: data initial address of the MODBUS relation.

Data Size: the Data Size value sets the maximum amount of data that a MODBUS relation can access from the initial address. Thus, in order to read a continuous range of addresses, it is necessary that all addresses are declared in a single relation. This field varies according to the configured type of MODBUS data.

Data Range: this field shows the user the memory address range used by the MODBUS relation.

ATTENTION

Differently from other application tasks, when a depuration mark in the MainTask is reached, the task of a MODBUS RTU Slave instance and any other MODBUS task will stop running at the moment that it tries to perform a writing in a memory area. It occurs in order to keep the consistency of the memory areas data while a MainTask is not running.

5.5.6.2. MODBUS Slave Protocol Configuration via Direct Representation (%Q)

To configure this protocol using Direct Representation (%Q), you must perform the following steps:

- Configure the general parameters of MODBUS slave protocol, such as: communication times, address and direct representation variables (%Q) to receive diagnostics and control relations.
- Add and configure MODBUS relations, specifying the MODBUS data type, direct representation variables (%Q) to receive/write the data and amount of data to communicate.

The descriptions of each setting are listed below, in this section.

5.5.6.2.1. General Parameters of MODBUS Slave Protocol – Configuration via Direct Representation (%Q)

The general parameters, found on the home screen of MODBUS protocol configuration (figure below), are defined as:

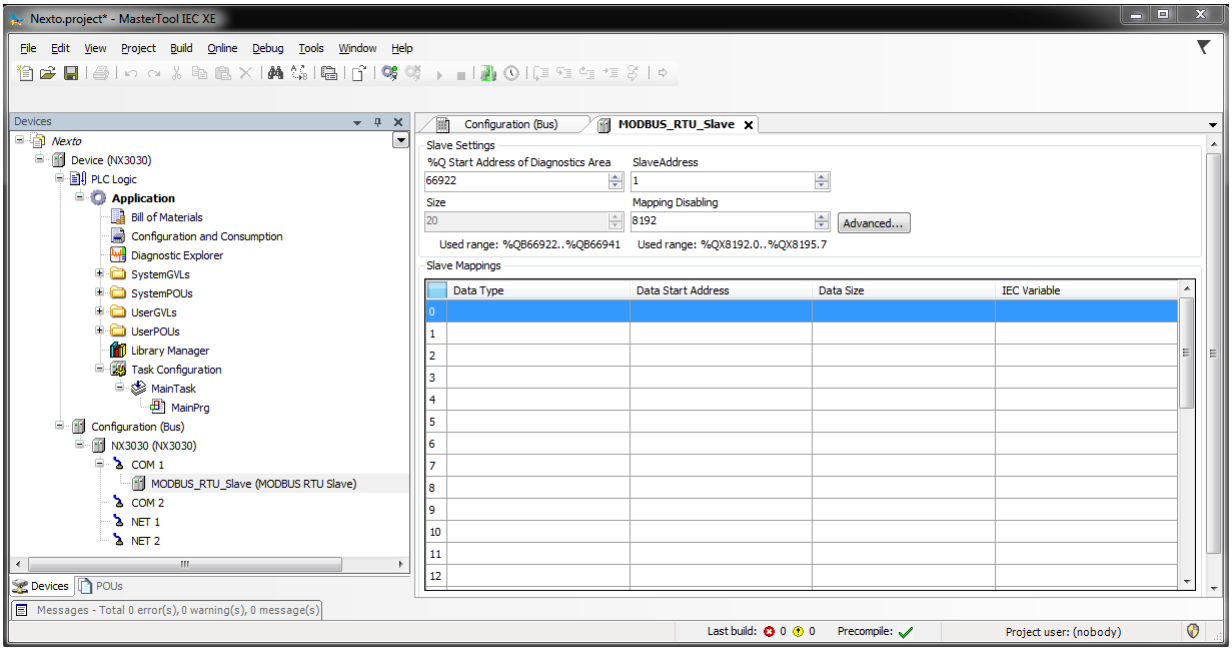


Figure 64: MODBUS RTU Slave Configuration Screen by Direct Representation

Address and direct representation variables (%Q) to control relations and diagnostics:

Configuration	Description	Default Value	Options
%Q Start Address of Diagnostics Area	Initial address of the diagnostic variables	-	0 to 2147483628
Size	Size of diagnostics area	-	Disabled for editing
Slave Address	MODBUS slave address	1	1 to 255
Mapping Disabling	Initial address used to disable MODBUS relations	-	0 to 2147483644

Table 103: Address and Direct Representation Variables Settings

Notes:

%Q Start Address of Diagnostics Area: this field is limited by the size of output variables addressable memory (%Q) of each CPU, which can be found in section Memory.

Slave Address: it is important to note that the Slave accepts requests broadcast, when the master sends a command with the address set to zero. Moreover, in accordance with standard MODBUS, the valid address range for slaves is 1 to 247. The addresses 248 to 255 are reserved.

Mapping Disabling: composed of 32 bits, used to disable, individually, the 32 MODBUS relations configured in *Slave Mappings* space. The relation is disabled when the corresponding bit is equal to 1, otherwise, the mapping is enabled. This field is limited by the size of output variables addressable memory (%Q) of each CPU, which can be found on Memory section.

Default Value: the factory default value cannot be set for the *%Q Start Address of Diagnostics Area* and *Mapping Disabling* fields, since the creation of a relation can be performed at any time on application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables (%Q), still unused.

The MODBUS Slave by Direct Representation protocol stops communicating while the CPU is in STOP or stopped at a breakpoint.

The MODBUS protocol diagnostics and commands are described in the Table 100.

The communication times of the MODBUS Slave protocol, found on the button *Advanced...* of the configuration screen, are described in Table 99.

5.5.6.2.2. Mappings Configuration – Configuration via Direct Representation (%Q)

The MODBUS relations settings, viewed in the figures below, follow the parameters described in table below:

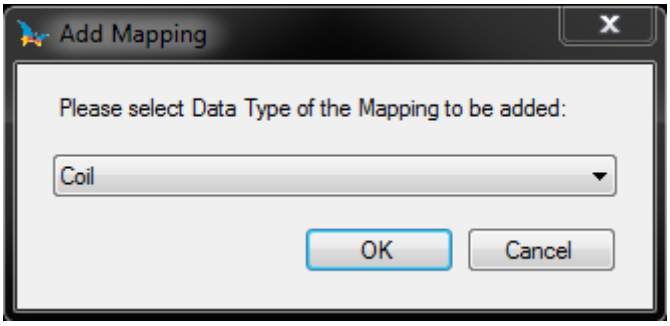


Figure 65: Adding MODBUS Relations

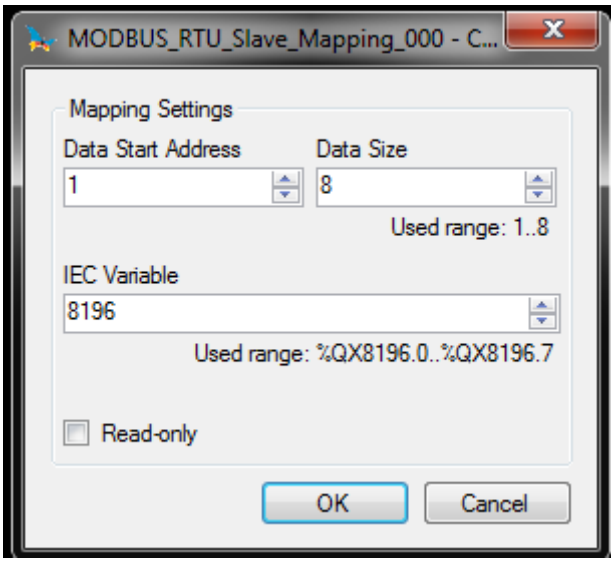


Figure 66: Configuring the MODBUS Relation

Configuration	Description	Default Value	Options
Data Type	MODBUS data type	Coil	Coil (1 bit) Holding Register (16 bits) Input Register (16 bits) Input Status (1 bit)
Data Start Address	Initial address of the MODBUS data	1	1 to 65536
Data Size	Number of MODBUS data	-	1 to 65536
IEC Variable	Initial address of variables (%Q)	-	0 to 2147483647
Read-only	Only allows reading	Disabled	Enabled or disabled

Table 104: Slave Mappings

Notes:

Options: the values written in the column *Options* may vary according with the configured MODBUS data.

Data Size: the value of *Data Size* defines the maximum amount of data that a MODBUS relation can access, from the initial address. Thus, to read a continuous address range, it is necessary that all addresses are declared in a single interface.

This field varies with the MODBUS data type configured, i.e. when selected *Coil* or *Input Status*, the Data Size field must be a multiple of eight. Also, the maximum amount must not exceed the size of output addressable memory and not assign the same values used in the application.

ATTENTION

When accessing the communication data memory is between devices with different endianism (Little-Endian and Big-Endian), inversion of the read/write data may occur. In this case, the user must adjust the data in the application.

IEC Variable: in case the MODBUS data type is *Coil* or *Input Status* (bit), the IEC variables initial address will be in the format *%QX10.1*. However, if the MODBUS data type is *Holding Register* or *Input Register* (16 bits), the IEC variables initial address will be in the format *%QW*. This field is limited by the memory size of the addressable output variables (%Q) from each CPU, which can be seen on Memory section.

Read-only: when enabled, it only allows the communication master to read the variable data. It does not allow the writing. This option is valid for the writing functions only.

Default Value: the default value cannot be defined for the *IEC Variable* field since the creation of a relation can be performed at any time on application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables (%Q), still unused. The default cannot be defined for the *Data Size* field as it will vary according to selected MODBUS data type.

In the previously defined relations, the maximum MODBUS data size can be 65535 (maximum value configured in the *Data Size* field). However, the request which arrives in the MODBUS RTU Slave must address a subgroup of this mapping and this group must have, at most, the data size depending on the function code which is defined below:

- Read Coils (FC 1): 2000
- Read Input Status (FC 2): 2000
- Read Holding Registers (FC 3): 125
- Read Input Registers (FC 4): 125
- Write Single Coil (FC 5): 1
- Write Single Holding register (FC 6): 1
- Force Multiple Coils (FC 15): 1968
- Write Holding Registers (FC 16): 123
- Mask Write Register (FC 22): 1
- Read/Write Holding Registers (FC 23):
  - Read: 121
  - Write: 121

ATTENTION

Differently from other application tasks, when a depuration mark in the MainTask is reached, the task of a Slave MODBUS RTU instance and any other MODBUS task will stop running at the moment that it tries to perform a writing in a memory area. It occurs in order to keep the consistency of the memory areas data while a MainTask is not running.

5.5.7. MODBUS Ethernet

The multi-master communication allows the Nexto CPUs to read or write MODBUS variables in other controllers or HMIs compatible with the MODBUS TCP protocol or MODBUS RTU via TCP. The Nexto CPU can, at the same time, be client and server in the same communication network, or even have more instances associated to the Ethernet interface. It does not matter if they are MODBUS TCP or MODBUS RTU via TCP, as described on Table 67.

The figure below represents some of the communication possibilities using the MODBUS TCP protocol simultaneously with the MODBUS RTU via TCP protocol.

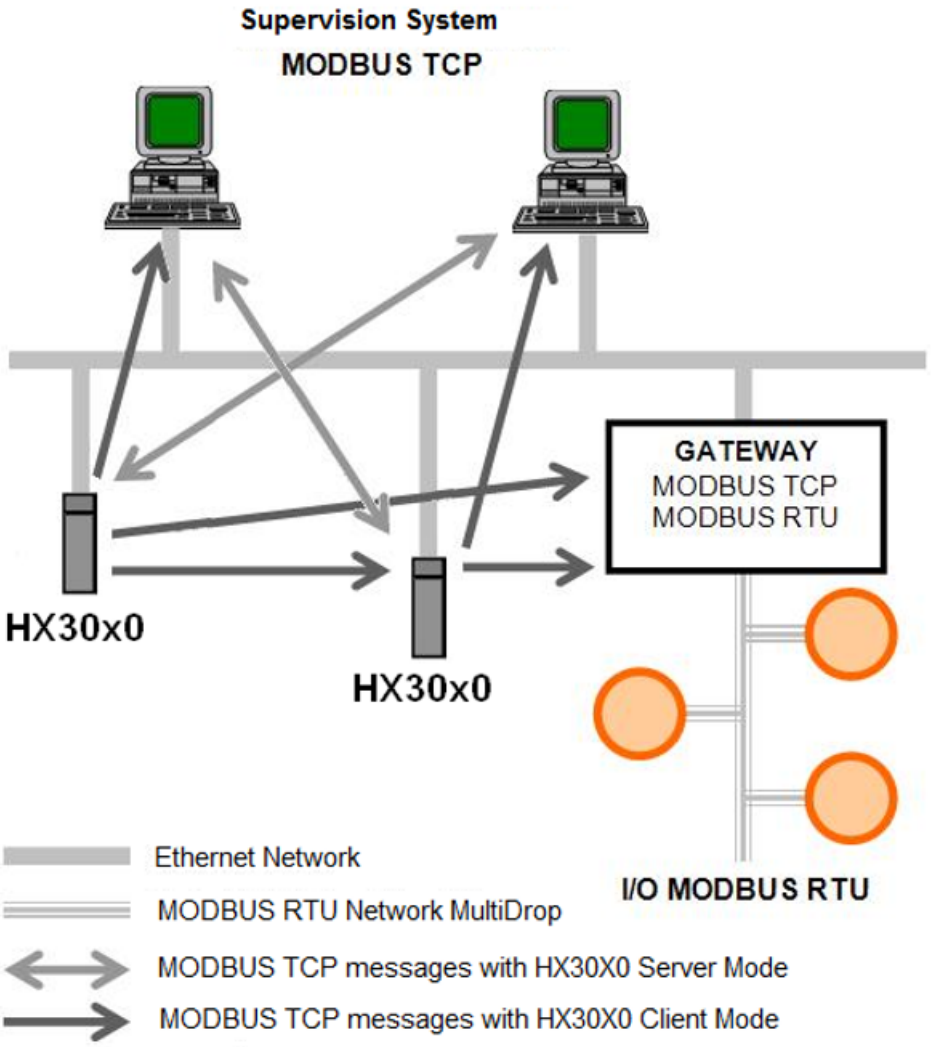


Figure 67: MODBUS TCP Communication Network

The association of MODBUS variables with CPU symbolic variables is made by the user through relations definition via MasterTool IEC XE configuration tool. It's possible to configure up to 32 relations for the server mode and up to 128 relations for the client mode. The relations in client mode, on the other hand, must respect the data maximum size of a MODBUS function: 125 registers (input registers or holding registers) or 2000 bits (coils or input status). This information is detailed in the description of each protocol.

All relations, in client mode or server mode, can be disabled through direct representation variables (%Q) identified as Disabling Variables by MasterTool IEC XE. The disabling may occur through general bits which affect all relations of an operation mode, or through specific bits, affecting specific relations.

For the server mode relations, IP addresses clusters can be defined with writing and reading allowance, called filters. This is made through the definition of an IP network address and of a subnet mask, resulting in a group of client IPs which can read and write in the relation variables. Reading/writing functions are filtered, in other words, they cannot be requested by any client, independent from the IP address. This information is detailed in the MODBUS Ethernet Server protocol.

When the MODBUS TCP protocol is used in the client mode, it's possible to use the multiple requests feature, with the same TCP connection to accelerate the communication with the servers. When this feature isn't desired or isn't supported by the server, it can be disabled (relation level action). It is important to emphasize that the maximum number of TCP connections between the client and server is 63. If some parameters are changed, inactive communications can be closed, which allows the opening of new connections.

The tables below bring, respectively, the complete list of data and MODBUS functions supported by the Nexto CPUs.

Data Type	Size [bits]	Description
Coil	1	Digital output that can be read or written.
Input Status	1	Digital input (read only).
Holding Register	16	Analog output that can be read or written.
Input Register	16	Analog input (read only).

Table 105: MODBUS data types supported by Nexto CPUs

Code		Description
DEC	**HEX**	
1	0x01	Read Coils (FC 01)
2	0x02	Read Input Status (FC 02)
3	0x03	Read Holding Registers (FC 03)
4	0x04	Read Input Registers (FC 04)
5	0x05	Write Single Coil (FC 05)
6	0x06	Write Single Holding Register (FC 06)
15	0x0F	Write Multiple Coils (FC 15)
16	0x10	Write Multiple Holding Registers (FC 16)
22	0x16	Mask Write Holding Register (FC 22)
23	0x17	Read/Write Multiple Holding Registers (FC 23)

Table 106: MODBUS Functions Supported by Nexto CPUs

Independent of the configuration mode, the steps to insert an instance of the protocol and configure the Ethernet interface are equal. The remaining configuration steps are described below for each modality.

- Add one or more instances of the MODBUS Ethernet client or server protocol to Ethernet channel. To perform this procedure, refer to the section Inserting a Protocol Instance.
- Configure the Ethernet interface. To perform this procedure, see section Ethernet Interfaces Configuration.

5.5.8. MODBUS Ethernet Client

This protocol is available for all Nexto Series CPUs on its Ethernet channels. When selecting this option at MasterTool IEC XE, the CPU becomes a MODBUS communication client, allowing the access to other devices with the same protocol, when it's in execution mode (*Run* Mode).

There are two ways to configure this protocol. The first one makes use of *direct representation (%Q)*, in which the variables are defined by your address. The second one, through *symbolic mapping*, where the variables are defined by your name.

The procedure to insert an instance of the protocol is found in detail in the MasterTool IEC XE User Manual – MU299609 or on Inserting a Protocol Instance section.

5.5.8.1. MODBUS Ethernet Client Configuration via Symbolic Mapping

To configure this protocol using *Symbolic Mapping*, it's necessary to execute the following steps:

- Configure the general parameters of MODBUS protocol client, with the Transmission Control Protocol (TCP) or RTU via TCP.
- Add and configure devices by setting IP address, port, address of the slave and time-out of communication (available on the Advanced Settings button of the device).
- Add and configure the MODBUS mappings, specifying the variable name, data type, data initial address, data size and variable that will receive the quality data.
- Add and configure the MODBUS request, specifying the desired function, the scan time of the request, the initial address (read/write), the size of the data (read/write), the variable that will receive the data quality and the variable responsible for disabling the request.

5.5.8.1.1. MODBUS Client Protocol General Parameters – Configuration via Symbolic Mapping

The general parameters, found on the MODBUS protocol configuration initial screen (figure below), are defined as:

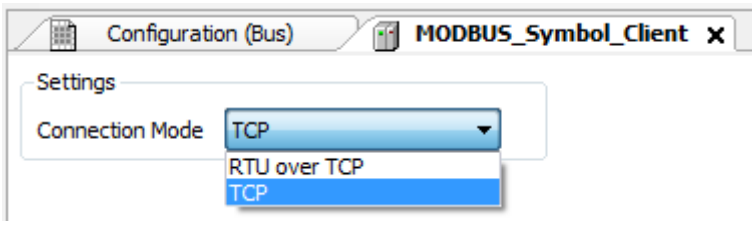


Figure 68: MODBUS Client General Parameters Configuration Screen

Configuration	Description	Default	Options
Connection Mode	Protocol selection	TCP	RTU via TCP TCP

Table 107: MODBUS Client General Configurations

The MODBUS Client protocol diagnostics and commands configured, either by symbolic mapping or direct representation, are stored in *T_DIAG_MODBUS_ETH_CLIENT_1* variables. For the direct representation mapping, they are also in 4 bytes and 8 words which are described in table below (where "n" is the configured value in the *%Q Start Address of Diagnostics Area*):

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_CLIENT_1.*	Size	Description
Diagnostic Bits:			
%QX(n).0	tDiag. bRunning	BIT	The client is in execution mode.
%QX(n).1	tDiag. bNotRunning	BIT	The client is not in execution mode (see bit bInterruptedByCommand).
%QX(n).2	tDiag. bInterruptedByCommand	BIT	The bit bNotRunning was enabled, as the client was interrupted by the user through command bits.
%QX(n).3	tDiag. bConfigFailure	BIT	Discontinued diagnostics.
%QX(n).4	tDiag. bRXFailure	BIT	Discontinued diagnostics.
%QX(n).5	tDiag. bTXFailure	BIT	Discontinued diagnostics.
%QX(n).6	tDiag. bModuleFailure	BIT	Indicates if there is failure in the module or the module is not present.
%QX(n).7	tDiag. bAllDevicesCommFailure	BIT	Indicates that all devices configured in the Client are in failure.
%QB(n+1)	byDiag_1_reserved	BYTE	Reserved.
Command bits, automatically initialized:			
%QX(n+2).0	tCommand. bStop	BIT	Stop client.

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS_ETH_CLIENT_1.*	Size	Description
%QX(n+2).1	tCommand. bRestart	BIT	Restart client.
%QX(n+2).2	tCommand. bResetCounter	BIT	Restart the diagnostic statistics (counters).
%QX(n+2).3	tCommand. bDiag_19_reserved	BIT	Reserved.
%QX(n+2).4	tCommand. bDiag_20_reserved	BIT	Reserved.
%QX(n+2).5	tCommand. bDiag_21_reserved	BIT	Reserved.
%QX(n+2).6	tCommand. bDiag_22_reserved	BIT	Reserved.
%QX(n+2).7	tCommand. bDiag_23_reserved	BIT	Reserved.
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
Communication Statistics:			
%QW(n+4)	tStat. wTXRequests	WORD	Counter of number of requests transmitted by the client (0 to 65535).
%QW(n+6)	tStat. wRXNormalResponses	WORD	Counter of normal answers received by the client (0 to 65535).
%QW(n+8)	tStat. wRXExceptionResponses	WORD	Counter of answers with exception code (0 to 65535).
%QW(n+10)	tStat. wRXIllegalResponses	WORD	Counter of illegal answers received by the client – invalid syntax, invalid CRC or not enough bytes received (0 to 65535).
%QW(n+12)	tStat. wDiag_12_reserved	WORD	Reserved.
%QW(n+14)	tStat. wDiag_14_reserved	WORD	Reserved.
%QW(n+16)	tStat. wDiag_16_reserved	WORD	Reserved.
%QW(n+18)	tStat. wDiag_18_Reserved	WORD	Reserved.

Table 108: MODBUS Client Protocol Diagnostics

Note:
Counters: all MODBUS TCP Client diagnostics counters return to zero when the limit value 65535 is exceeded.

5.5.8.1.2. Device Configuration – Configuration via Symbolic Mapping

The devices configuration, shown on figure below, follows the following parameters:

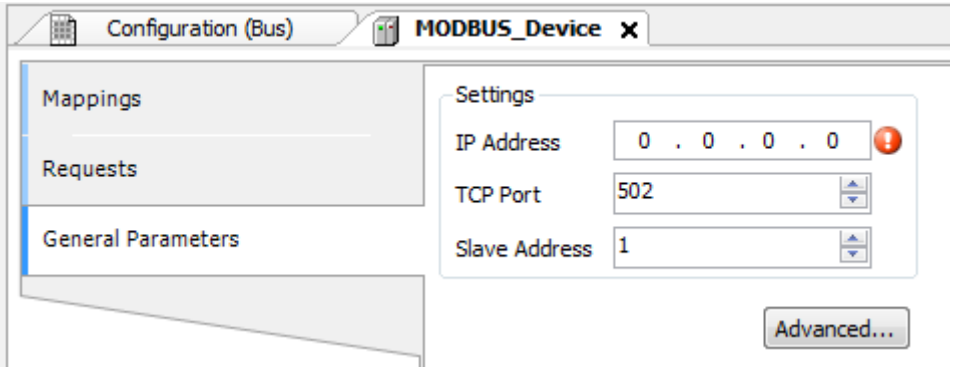


Figure 69: Device General Parameters Settings

Configuration	Description	Default	Options
IP Address	Server IP address	0.0.0.0	1.0.0.1 to 223.255.255.255
TCP Port	TCP port	502	2 to 65534
Slave Address	MODBUS Slave address	1	0 to 255

Table 109: MODBUS Client General Configurations

Notes:

IP Address: IP address of Modbus Server Device.

TCP Port: if there are multiple instances of the protocol added in a single Ethernet interface, different TCP ports must be selected for each instance. Some TCP ports, among the possibilities mentioned above, are reserved and therefore cannot be used. See table Reserved TCP/UDP ports.

Slave address: according to the MODBUS standard, the valid address range for slaves is 0 to 247, where addresses 248 to 255 are reserved. When the master sends a command of writing with the address set to zero, it is performing broadcast requests on the network.

The parameters in the advanced settings of the MODBUS Client device, found on the button *Advanced...* in the *General Parameters* tab are divided into: *Maximum Simultaneous Requests*, *Communication Time-out*, *Mode* of Connection Time-out and *Inactive Time*.

Configuration	Description	Default	Options
Maximum Simultaneous Request	Number of simultaneous request the client can ask from the server	1	1 to 8
Communication Time-out (ms)	Application level time-out in ms	3000	10 to 65535
Mode	Defines when the connection with the server finished by the client	Connection is closed after an inactive time of (s): 10 to 3600.	Connection is closed after a time-out. Connection is closed at the end of each communication. Connection is closed after an inactive time of (s): 10 to 3600.
Inactive Time (s)	Inactivity time	10	3600

Table 110: MODBUS Client Advanced Configurations

Notes:

Maximum Simultaneous Requests: it is used with a high scan cycle. This parameter is fixed in 1 (not editable) when the configured protocol is MODBUS RTU over TCP.

Communication Time-out: the Communication time-out is the time that the client will wait for a server response to the request. For a MODBUS Client device, two variables of the system must be considered: the time the server takes to process a request and the response sending delay in case it is set in the server. It is recommended that the time-out is equal or higher than twice the sum of these parameters. For further information, check Communication Performance section.

Mode: defines when the connection with the server is finished by the client. Below follows the available options:

- Connection is closed after a time-out or Connection is never closed in normal situations: Those options presents the same behavior of Client, close the connection due non response of a request by the Server before reaching the Communication Time-out.
- Connection is closed at the end of each communication: The connection is closed by the Client after finish each request.
- Connection is closed after an Inactive Time: The connection will be closed by the Client if it reach the Inactive Time without performing a request to the Server.

Inactive Time: inactivity connection time.

5.5.8.1.3. Mappings Configuration – Configuration via Symbolic Mapping

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:

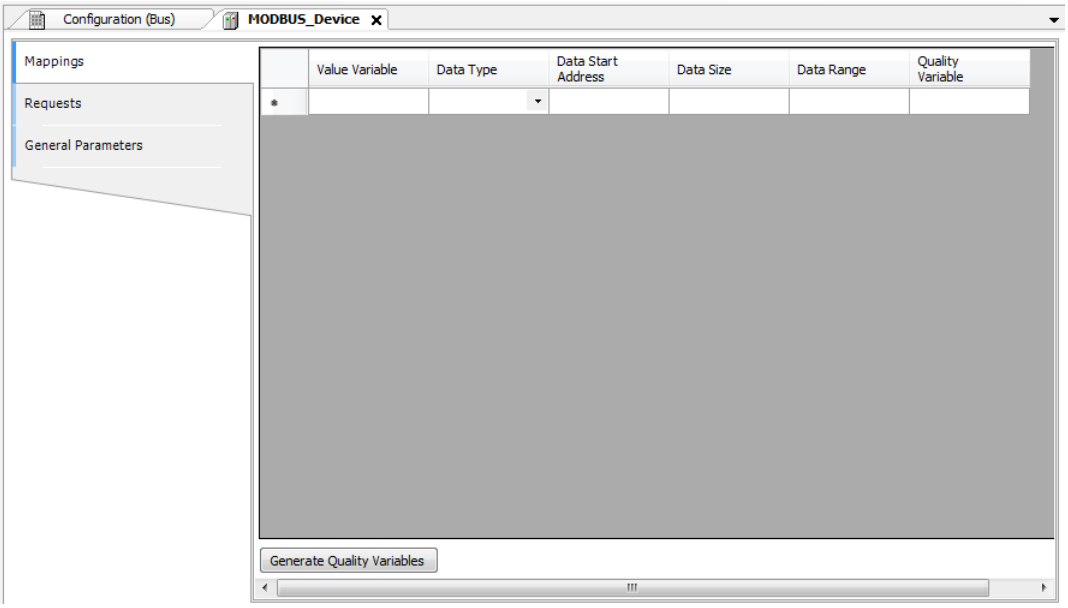


Figure 70: MODBUS Data Type

Configuration	Description	Default	Options
Value Variable	Symbolic variable name	-	Name of a variable declared in a program or GVL
Data Type	MODBUS data type	-	Coil - Write (1 bit) Coil - Read (1 bit) Holding Register - Write (16 bits) Holding Register - Read (16 bits) Holding Register – Mask AND (16 bits) Holding Register – Mask OR (16 bits) Input Register (16 bits) Input Status (1 bit)
Data Start Address	Initial address of the MODBUS data	-	1 to 65536
Data Size	Size of the MODBUS data	-	1 to 65536
Data Range	The address range of configured data	-	-

Table 111: MODBUS Mappings Settings

Notes:

Value Variable: this field is used to specify a symbolic variable in MODBUS relation.

Data type: this field is used to specify the data type used in the MODBUS relation.

Data Type	Size [bits]	Description
Coil - Write	1	Writing digital output.
Coil - Read	1	Reading digital output.
Holding Register - Write	16	Writing analog output.
Holding Register - Read	16	Reading analog output.
Holding Register - Mask AND	16	Analog output which can be read or written with AND mask.
Holding Register - Mask OR	16	Analog output which can be read or written with OR mask.
Input Register	16	Analog input which can be only read.
Input Status	1	Digital input which can be only read.

Table 112: Data Types Supported in MODBUS

Data Start Address: Data initial address of a MODBUS mapping.

Data Size: The size value specifies the maximum amount of data that a MODBUS interface can access, from the initial address. Thus, to read a continuous address range, it is necessary that all addresses are declared in a single interface. This field varies with the MODBUS data type configured.

Data Range: This field shows to the user the memory address range used by the MODBUS interface.

5.5.8.1.4. Requests Configuration – Configuration via Symbolic Mapping

The configuration of the MODBUS requests, viewed in figure below, follow the parameters described in table below:

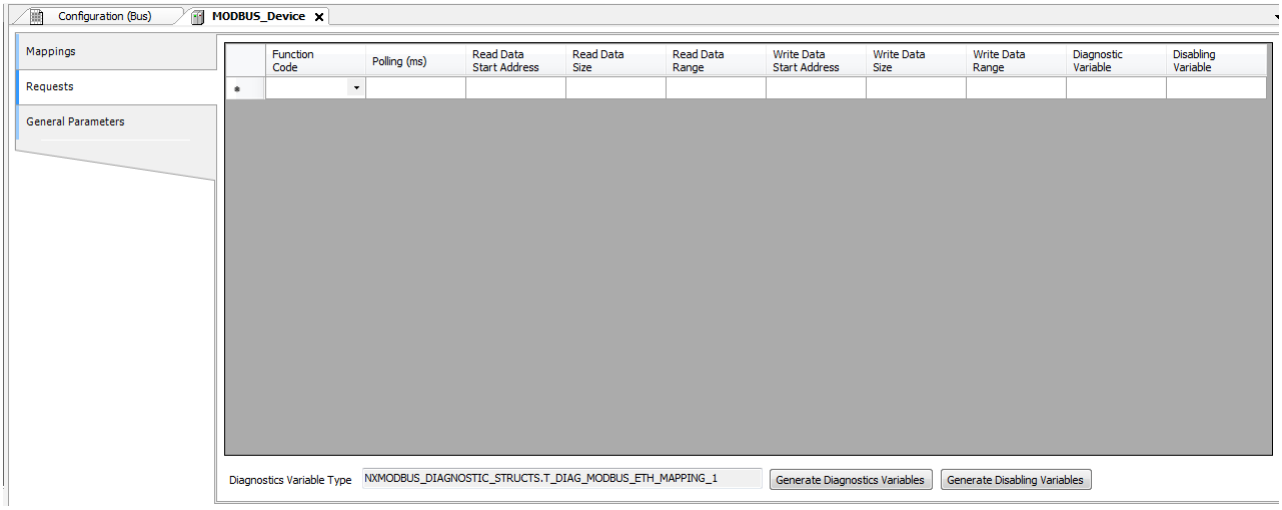


Figure 71: MODBUS Data Request Screen

Configuration	Description	Default Value	Options
Function Code	MODBUS function type	-	01 – Read Coils 02 – Read Input Status 03 – Read Holding Registers 04 – Read Input Registers 05 – Write Single Coil 06 – Write Single Register 15 – Write Multiple Coils 16 – Write Multiple Registers 22 – Mask Write Register 23 – Read/Write Multiple Registers
Polling (ms)	Communication period (ms)	100	0 to 3600000
Read Data Start Address	Initial address of the MODBUS read data	-	1 to 65536
Read Data Size	Size of MODBUS Read data	-	Depends on the function used
Read Data Range	MODBUS Read data address range	-	0 to 2147483646
Write Data Start Address	Initial address of the MODBUS write data	-	1 to 65536
Write Data Size	Size of MODBUS Write data	-	Depends on the function used
Write Data Range	MODBUS Write data address range	-	0 to 2147483647
Diagnostic Variable	Diagnostic variable name	-	Name of a variable declared in a program or GVL

Configuration	Description	Default Value	Options
Disabling Variable	Variable used to disable MODBUS relation	-	Field for symbolic variable used to disable, individually, MODBUS requests configured. This variable must be of type BOOL. The variable can be simple or array element and can be in structures.

Table 113: MODBUS Relations Configuration

Notes:

Setting: the number of factory default settings and the values for the column Options may vary according to the data type and MODBUS function (FC).

Function Code: MODBUS (FC) functions available are the following:

Code		Description
DEC	HEX	
1	0x01	Read Coils (FC 01)
2	0x02	Read Input Status (FC 02)
3	0x03	Read Holding Registers (FC 03)
4	0x04	Read Input Registers (FC 04)
5	0x05	Write Single Coil (FC 05)
6	0x06	Write Single Holding Register (FC 06)
15	0x0F	Write Multiple Coils (FC 15)
16	0x10	Write Multiple Holding Registers (FC 16)
22	0x16	Mask Write Holding Register (FC 22)
23	0x17	Read/Write Multiple Holding Registers (FC 23)

Table 114: MODBUS Functions Supported by Nexto CPUs

Polling: this parameter indicates how often the communication set for this request must be performed. By the end of a communication will be awaited a time equal to the value configured in the field polling and after that, a new communication will be executed.

Read Data Start Address: field for the initial address of the MODBUS read data.

Read Data Size: the minimum value for the read data size is 1 and the maximum value depends on the MODBUS function (FC) used as below:

- Read Coils (FC 01): 2000
- Read Input Status (FC 02): 2000
- Read Holding Registers (FC 03): 125
- Read Input Registers (FC 04): 125
- Read/Write Multiple Registers (FC 23): 121

Read Data Range: this field shows the MODBUS read data range configured for each request. The initial address, along with the read data size will result in the range of read data for each request.

Write Data Start Address: field for the initial address of the MODBUS write data.

Write Data Size: the minimum value for the write data size is 1 and the maximum value depends on the MODBUS function (FC) used as below:

- Write Single Coil (FC 05): 1
- Write Single Register (FC 06): 1
- Write Multiple Coils (FC 15): 1968

- Write Multiple Registers (FC 16): 123
- Mask Write Register (FC 22): 1
- Read/Write Multiple Registers (FC 23): 121

Write Data Range: this field shows the MODBUS write data range configured for each request. The initial address, along with the read data size will result in the range of write data for each request.

Diagnostic Variable: The MODBUS request diagnostics configured by symbolic mapping or by direct representation, are stored in variables of type *T_DIAG_MODBUS_RTU_MAPPING_1* for Master devices and *T_DIAG_MODBUS_ETH_CLIENT_1* for Client devices and the mapping by direct representation are in 4-byte and 2-word, which are described in Table 93 ("n" is the value configured in the *%Q Start Address of Diagnostics Area* field).

Direct Representation Variable	**Diagnostic Variable T_DIAG_MODBUS _ETH_MAPPING_1.***	**Size**	**Description**
Communication Status Bits:			
%QX(n).0	byStatus. bCommIdle	BIT	Communication idle (waiting to be executed).
%QX(n).1	byStatus. bCommExecuting	BIT	Active communication.
%QX(n).2	byStatus. bCommPostponed	BIT	Communication deferred, because the maximum number of concurrent requests was reached. Deferred communications will be carried out in the same sequence in which they were ordered to avoid indeterminacy. The time spent in this State is not counted for the purposes of timeout. The bCommIdle and bCommExecuting bits are false when the bCommPostponed bit is true.
%QX(n).3	byStatus. bCommDisabled	BIT	Communication disabled. The bCommIdle bit is restarted in this condition.
%QX(n).4	byStatus. bCommOk	BIT	Communication terminated previously was held successfully.
%QX(n).5	byStatus. bCommError	BIT	Communication terminated previously had an error. Check error code.
%QX(n).6	byStatus. bCommAborted	BIT	Previously terminated communication was interrupted due to connection failure.
%QX(n).7	byStatus. bDiag_7_reserved	BIT	Reserved.
Last error code (enabled when bCommError = true):			
%QB(n+1)	eLastErrorCode	MASTER_ERROR _CODE (BYTE)	Informs the possible cause of the last error in the MODBUS mapping. Consult Table 116 for further details.
Last exception code received by master:			
%QB(n+2)	eLastExceptionCode	MODBUS _EXCEPTION (BYTE)	NO_EXCEPTION (0) FUNCTION_NOT_SUPPORTED (1) MAPPING_NOT_FOUND (2) ILLEGAL_VALUE (3) ACCESS_DENIED (128)* MAPPING_DISABLED (129)* IGNORE_FRAME (255)*

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_MAPPING_1.*	Size	Description
Communication statistics:			
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
%QW(n+4)	wCommCounter	WORD	Communications counter terminated, with or without errors. The user can test when communication has finished testing the variation of this counter. When the value 65535 is reached, the counter returns to zero.
%QW(n+6)	wCommErrorCounter	WORD	Communications counter terminated with errors. When the value 65535 is reached, the counter returns to zero.

Table 115: MODBUS Client Relations Diagnostics

Notes:

Exception Codes: the exception codes show in this filed is the server returned values. The definitions of the exception codes 128, 129 and 255 are valid only with Altus slaves. For slaves from other manufacturers these exception codes can have different meanings.

Disabling Variable: field for the variable used to disable MODBUS requests individually configured within requests. The request is disabled when the variable, corresponding to the request, is equal to 1, otherwise the request is enabled.

Last Error Code: The codes for the possible situations that cause an error in the MODBUS communication can be consulted below:

Code	Enumerable	Description
1	ERR_EXCEPTION	Reply is in an exception code (see eLastExceptionCode = Exception Code).
2	ERR_CRC	Reply with invalid CRC.
3	ERR_ADDRESS	MODBUS address not found. The address that replied the request was different than expected.
4	ERR_FUNCTION	Invalid function code. The reply's function code was different than expected.
5	ERR_FRAME_DATA_COUNT	The amount of data in the reply was different than expected.
7	ERR_NOT_ECHO	The reply is not an echo of the request (FC 05 and 06).
8	ERR_REFERENCE_NUMBER	Invalid reference number (FC 15 and 16).
9	ERR_INVALID_FRAME_SIZE	Reply shorter than expected.
20	ERR_CONNECTION	Error while establishing connection.
21	ERR_SEND	Error during transmission stage.
22	ERR_RECEIVE	Error during reception stage.
40	ERR_CONNECTION_TIMEOUT	Application level time-out during connection.
41	ERR_SEND_TIMEOUT	Application level time-out during transmission.
42	ERR_RECEIVE_TIMEOUT	Application level time-out while waiting for reply.
43	ERR_CTS_OFF_TIMEOUT	Time-out while waiting CTS = false in transmission.
44	ERR_CTS_ON_TIMEOUT	Time-out while waiting CTS = true in transmission.
128	NO_ERROR	No error since startup.

Table 116: MODBUS Relations Error Codes

ATTENTION

Unlike other tasks of an application, when a mark is reached at MainTask debugging, the MODBUS Ethernet Client instance task or any other MODBUS task will stop being executed at the moment it tries to write in the memory area. This occurs in order to maintain data consistency of memory areas while MainTask is not running.

5.5.8.2. MODBUS Ethernet Client configuration via Direct Representation (%Q)

To configure this protocol using direct representation (%Q), the following steps must be performed:

- Configure the general parameters of the MODBUS protocol, such as: communication times and direct representation variables (%Q) to receive diagnostics.
- Add and configure devices by setting address, direct representation variables (%Q) to disable the relations, communication time-outs, etc.
- Add and configure MODBUS relations, specifying the data type and MODBUS function, time-outs, direct representation variables (%Q) to receive diagnostics of the relation and other to receive/write the data, amount of data to be transmitted and relation polling.

The descriptions of each configuration are listed below in this section.

5.5.8.2.1. General parameters of MODBUS Protocol Client - configuration for Direct Representation (%Q)

The General parameters, found on the home screen of MODBUS protocol configuration (figure below), are defined as:

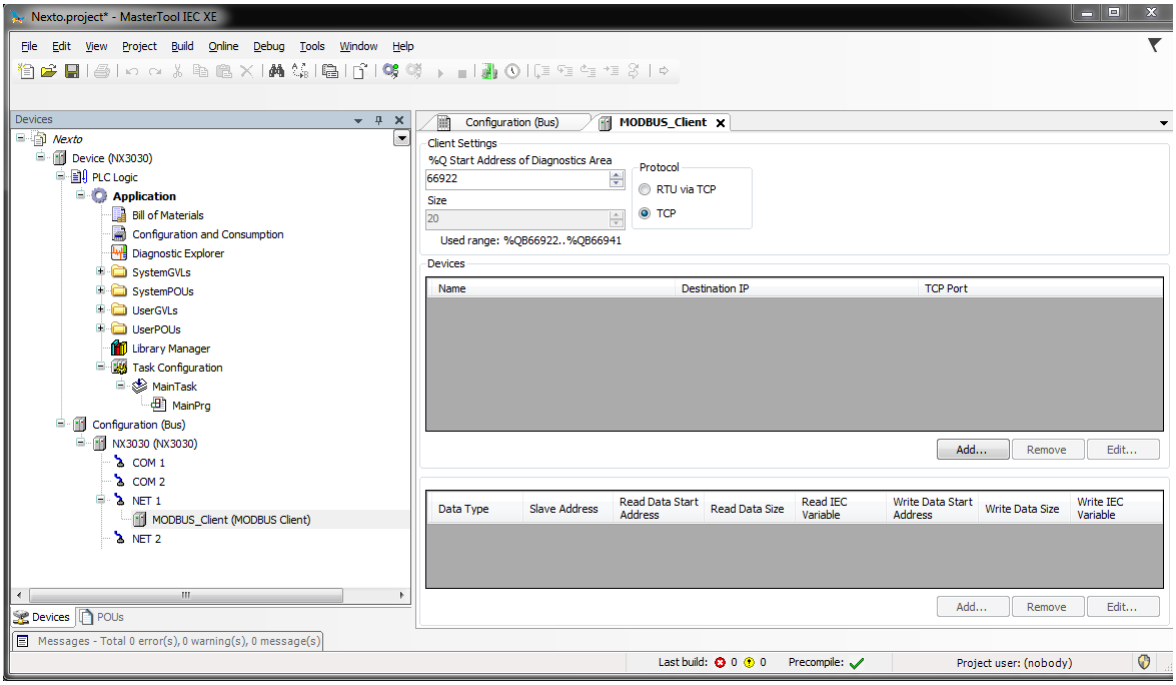


Figure 72: MODBUS Client Setup Screen

Protocol selection and direct representation variables (%Q) for diagnostics:

Setting	Description	Default Value	Options
%Q Start Address of Diagnostics Area	Initial address of the diagnostic variables	-	0 to 2147483628
Size	Size of diagnostics	20	Disabled for editing
Protocol	Protocol selection	TCP	RTU via TCP TCP

Table 117: MODBUS Client settings

Notes:

%Q Start Address of Diagnostics Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in section Memory.

Default Value: the default value cannot be defined for the *%Q Start Address of Diagnostics Area* field since the creation of a protocol instance can be made at any moment within the application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables (%Q), still unused.

The diagnostics and MODBUS commands are described in Table 108.

5.5.8.2.2. *Device Configuration – Configuration via Direct Representation (%Q)*

The configuration of the devices, viewed in figure below, comprises the following parameters:

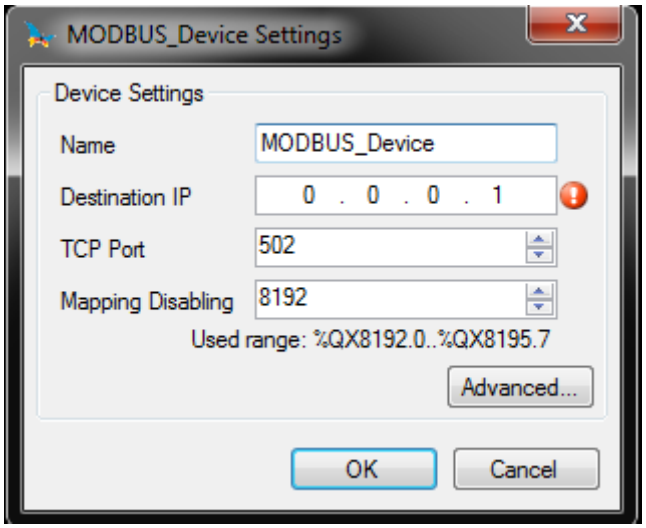


Figure 73: Configuring MODBUS Client

Configuration	Description	Factory default	Options
Name	Name of the instance	MODBUS_Device	Identifier, according to IEC 61131-3
Destination IP	IP address of the server	0. 0. 0.1	1.0.0.1 to 223.255.255.255
TCP Port	TCP Port	502	2 to 65534
Mapping Disabling	Initial address used to disable MODBUS relations	-	Any address of the %Q area, limited by the CPU model

Table 118: Configuration of Client Devices

Notes:

Instance Name: this field is the identifier of the device, which is checked according to IEC 61131-3, i.e. it does not allow spaces, special characters and starting with numeral character. It is limited to 24 characters.

TCP Port: if there are multiple instances of the protocol added in a single Ethernet interface, different TCP ports must be selected for each instance. Some TCP ports, among the possibilities mentioned above, are reserved and therefore cannot be used. See table Reserved TCP/UDP ports.

Mapping Disabling: composed of 32 bits, it is used to disable, individually, the 32 MODBUS relations configured in *Device Mappings* space. The relation is disabled when the corresponding bit is equal to 1, otherwise, the mapping is enabled. This field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in section Memory.

Default Value: factory default cannot be set for the *Mapping Disabling* field, since the creation of a protocol instance can be made at any moment within the application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables (%Q), still unused.

Communication Time-out: the settings present on the button *Advanced...* on the TCP connection, are described in the notes of the section Device Configuration – Configuration via Symbolic Mapping.

5.5.8.2.3. Mapping Configuration – Configuration via Direct Representation (%Q)

The MODBUS relations settings, viewed in the figures below, follow the parameters described in table below:

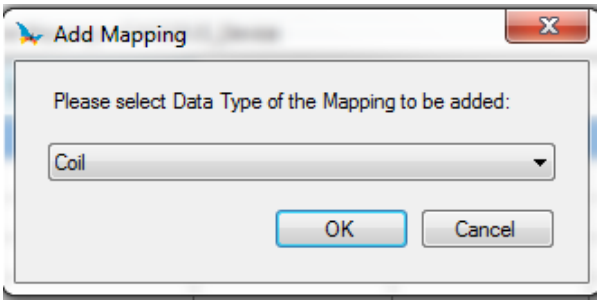


Figure 74: MODBUS Data Type

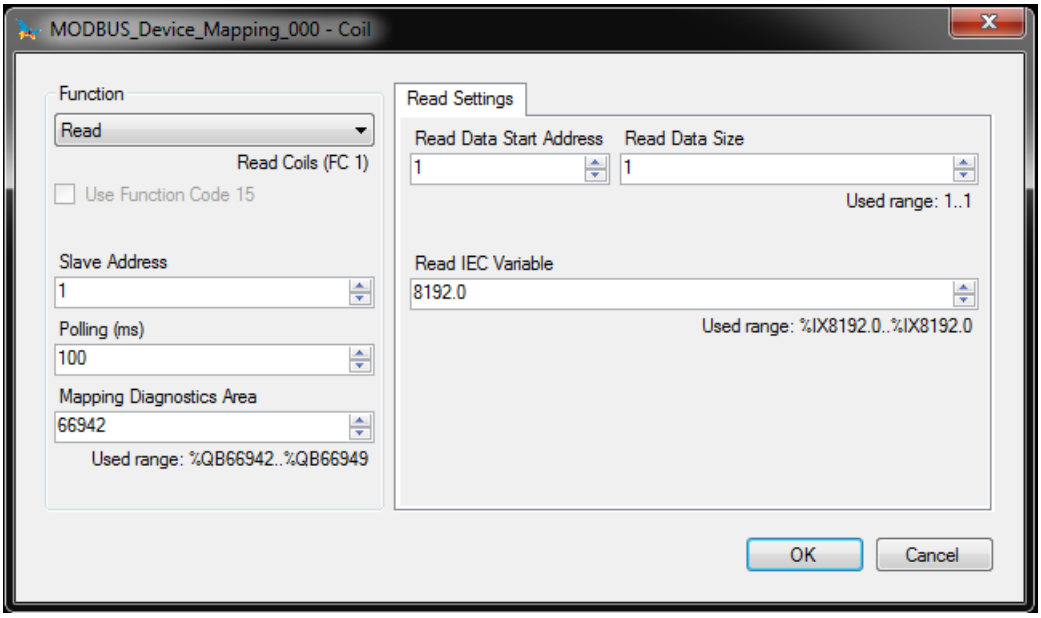


Figure 75: MODBUS Function

In table below, the number of factory default settings and the values for the column Options, may vary according to the data type and MODBUS function (FC).

Configuration	Description	Default Value	Options
Function	MODBUS function type	Read	Read Write Read/Write Mask Write
Slave Address	MODBUS slave address	1	0 to 255
Polling (ms)	Period of communication (ms)	100	0 to 3600000
Mapping Diagnostics Area	Starting address of MODBUS interface diagnostics	-	0 to 2147483640
Read Data Start Address	Starting address of the read MODBUS data	1	1 to 65536
Read Data Size	Number of read MODBUS data	-	Depends on the function used
Read IEC Variable	Starting address of the read variables (%I)	-	0 to 2147483647
Write Data Start Address	Starting address of MODBUS writing data	1	1 to 65536
Write Data Size	Number of MODBUS writing data	-	Depends on the function used
Write IEC Variable	Starting address of the write variables (%Q)	-	0 to 2147483647
Mask Write IEC Variables	Starting address of variables for write mask (%Q)	-	0 to 2147483644

Table 119: Device Mapping

Notes:

Device Mappings Table: the number of settings and values described in the column Options may vary according to the data type and MODBUS function.

Slave Address: typically, the address 0 is used when the server is a MODBUS RTU or MODBUS RTU via TCP Gateway, and the same broadcasts the request to all network devices. When the address 0 is used, the client doesn't waits for a response and its use serves only to written commands. Moreover, in accordance with MODBUS standard, the valid address range for slaves is 0 to 247, and addresses 248 to 255 are reserved.

Polling: this parameter indicates how often the communication set for this relation must be executed. At the end of communication will be awaited a time equal to the configured polling and after, will be performed a new communication as soon as possible.

Mapping Diagnostic Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in the section Memory . The configured MODBUS relations diagnostics are described in Table 93.

Size of the Read and Write Data: details of the size of the data supported by each function are described in the notes of Requests Configuration – Symbolic Mapping Settings section.

ATTENTION

When accessing the communication data memory is between devices with different endianism (Little-Endian and Big-Endian), inversion of the read/write data may occur. In this case, the user must adjust the data in the application.

Read IEC Variable: in case the MODBUS data type is *Coil* or *Input Status* (bit), the IEC variables initial address will be in the format *%IX10.1*. However, if the MODBUS data type is *Holding Register* or *Input Register* (16 bits), the IEC variables initial address will be in the format *%IW*. This field is limited by the memory size of the addressable input variables *(%I)* from each CPU, which can be seen on Memory section.

Write IEC Variable: in case the MODBUS data type is *Coil* (bit), the IEC variables initial address will be in the format *%QX10.1*. However, if the MODBUS data type is *Holding Register* (16 bits), the IEC variables initial address will be in the format *%QW*. This field is limited by the memory size of the addressable output variables *(%Q)* from each CPU, which can be seen on Memory section.

Write Mask of IEC Variables: the *Mask Write Register* function (FC 22) employs a logic between the value already written and the two words that are configured in this field using *%QW(0)* for the AND mask and *%QW(2)* for the OR mask; allowing the user to handle the word. This field is limited by the size of output variables addressable memory *(%Q)* of each CPU, which can be found in the section Memory.

Default Value: the factory default value cannot be set for the *Mapping Diagnostics Area*, *Read IEC Variable*, *Write IEC Variable* and *Mask Write IEC Variables* fields, since the creation of a relation can be performed at any time on application development. The MasterTool IEC XE software itself allocate a value from the range of direct representation output variables *(%Q)*, still unused. Factory default cannot be set to the *Read/Write Data Size* fields, as they will vary according to the MODBUS data type selected.

ATTENTION

Unlike other tasks of an application, when a mark is reached at MainTask debugging, the MODBUS Ethernet Client instance task or any other MODBUS task will stop being executed at the moment it tries to write in the memory area. This occurs in order to maintain data consistency of memory areas while MainTask is not running.

5.5.8.3. MODBUS Client Relation Start in Acyclic Form

To start a MODBUS Client relation in acyclic form, it is suggested the following method which can be implemented in a simple way in the user application program:

- Define the maximum polling time for the relations;
- Keep the relation normally disabled;
- Enable the relation at the moment the execution is desired;
- Wait for the confirmation of the relation execution finishing and, at this moment, disable it again.

5.5.9. MODBUS Ethernet Server

This protocol is available for all Nexto Series CPUs on its Ethernet channels. When selecting this option at MasterTool IEC XE, the CPU becomes a MODBUS communication server, allowing the connection with MODBUS client devices. This protocol is only available when the CPU is in execution mode (*Run* Mode).

There are two ways to configure this protocol. The first one makes use of *direct representation (%Q)*, in which the variables are defined by your address. The second one, through *symbolic mapping*, where the variables are defined by your name.

The procedure to insert an instance of the protocol is found in detail in the MasterTool IEC XE User Manual – MU299609.

5.5.9.1. MODBUS Server Ethernet Protocol Configuration for Symbolic Mapping

To configure this protocol using *Symbolic Mappings*, it is necessary to execute the following steps:

- Configure the MODBUS server protocol general parameters, as: TCP port, protocol selection, IP filters for Reading and Writing (available at the Filters Configuration button) and communication times (available at the Server Advanced Configurations button).
- Add and configure MODBUS mappings, specifying the variable name, data type, data initial address and data size.

The description of each configuration is related ahead in this section.

5.5.9.1.1. MODBUS Server Protocol General Parameters – Configuration via Symbolic Mapping

The general parameters, found on the MODBUS protocol initial screen (figure below), are defined as.

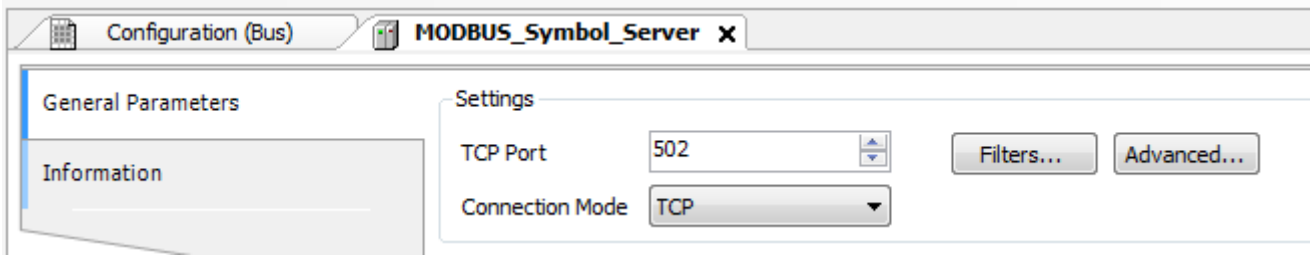


Figure 76: MODBUS Server General Parameters Configuration Screen

Configuration	Description	Default	Options
TCP Port	TCP port	502	2 to 65534
Connection Mode	Protocol selection	TCP	RTU via TCP TCP

Table 120: MODBUS Server General Configurations

Notes:

TCP Port: if there are multiple instances of the protocol added in a single Ethernet interface, different TCP ports must be selected for each instance. Some TCP ports, among the possibilities mentioned above, are reserved and therefore cannot be used. See table Reserved TCP/UDP ports.

The settings present on the *Filters...* button, described in table below, are relative to the TCP communication filters:

Configuration	Description	Default Value	Options
Write Filter IP Address	Specifies a range of IPs with write access in the variables declared in the MODBUS interface.	0.0.0.0	0.0.0.0 to 255.255.255.255
Write Filter Mask	Specifies the subnet mask in conjunction with the IP filter parameter for writing.	0.0.0.0	0.0.0.0 to 255.255.255.255

Configuration	Description	Default Value	Options
Read Filter IP Address	Specifies a range of IPs with read access in the variables declared in the MODBUS interface.	0.0.0.0	0.0.0.0 to 255.255.255.255
Read Filter Mask	Specifies the subnet mask in conjunction with the IP filter parameter for reading.	0.0.0.0	0.0.0.0 to 255.255.255.255

Table 121: IP Filters

Note:

Filters: filters are used to establish a range of IP addresses that have write or read access to MODBUS relations, being individually configured. The permission criteria is accomplished through a logical AND operation between the Write Filter Mask and the IP address of the client. If the result is the same as the Write Filter IP Address, the client is entitled to write. For example, if the Write Filter IP Address = 192.168.15.0 and the Write Filter Mask = 255.255.255.0, then only customers with IP address = 192.168.15.x shall be entitled. The same procedure is applied in the Read Filter parameters to define the read rights.

The communication times of the MODBUS server protocol, found on the *Advanced...* button of the configuration screen, are divided into: *Task Cycle* and *Connection Inactivity Time-out*.

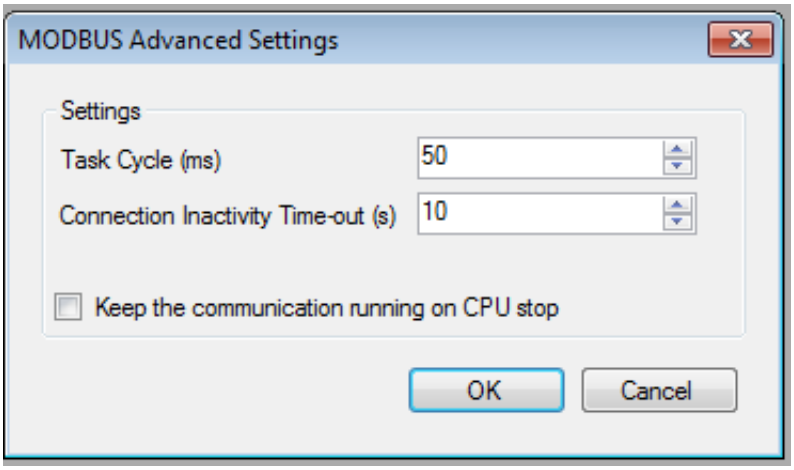


Figure 77: MODBUS Server Advanced Settings Configuration Screen

Configuration	Description	Default Value	Options
Task Cycle (ms)	Time for the instance execution within the cycle, without considering its own execution time	50	5 to 100
Connection Inactivity Time-out (s)	Maximum idle time between client and server before the connection is closed by the server	10	1 to 3600
Keep the communication running on CPU stop.	Enable the MODBUS Symbol Slave to run while the CPU is in STOP or after a breakpoint	Unmarked	Marked or Unmarked

Table 122: MODBUS Server Advanced Configurations

Notes:

Task Cycle: the user has to be careful when changing this parameter as it interferes directly in the answer time, data volume for scanning and mainly in the CPU resources balance between communications and other tasks.

Connection Inactivity Time-out: this parameter was created in order to avoid that the maximum quantity of TCP connections is reached, imagining that inactive connections remain open on account of the most different problems. It indicates how long a connection (client or server) can remain open without being used (without exchanging communication messages). If the specified time is not reached, the connection is closed releasing an input in the connection table.

5.5.9.1.2. MODBUS Server Diagnostics – Configuration via Symbolic Mapping

The diagnostics and commands of the MODBUS server protocol configured, either by symbolic mapping or by direct representation, are stored in variables of type *T_DIAG_MODBUS_ETH_SERVER_1* and the mapping by direct representation are in 4-byte and 8-word, which are described in table below (n is the value configured in the *%Q Start Address of Diagnostics Area* field):

Direct Representation Variable	**Diagnostic Variable T_DIAG_MODBUS _ETH_SERVER_1 .***	**Size**	**Description**
Diagnostic bits:			
%QX(n).0	tDiag. bRunning	BIT	The server is running.
%QX(n).1	tDiag. bNotRunning	BIT	The server is not running (see bit bInterruptedByCommand).
%QX(n).2	tDiag. bInterruptedByCommand	BIT	The bit bNotRunning was enabled, because the server was interrupted by the user through the command bit.
%QX(n).3	tDiag. bConfigFailure	BIT	Discontinued diagnostic.
%QX(n).4	tDiag. bRXFailure	BIT	Discontinued diagnostic.
%QX(n).5	tDiag. bTXFailure	BIT	Discontinued diagnostic.
%QX(n).6	tDiag. bModuleFailure	BIT	Discontinued diagnostic.
%QX(n).7	tDiag. bDiag_7_reserved	BIT	Reserved.
%QB(n+1)	byDiag_1_reserved	BYTE	Reserved.
Command bits, restarted automatically:			
%QX(n+2).0	tCommand. bStop	BIT	Stop the server.
%QX(n+2).1	tCommand. bRestart	BIT	Restart the server.
%QX(n+2).2	tCommand. bResetCounter	BIT	Reset diagnostics statistics (counters).
%QX(n+2).3	tCommand. bDiag_19_reserved	BIT	Reserved.
%QX(n+2).4	tCommand. bDiag_20_reserved	BIT	Reserved.
%QX(n+2).5	tCommand. bDiag_21_reserved	BIT	Reserved.

Direct Representation Variable	Diagnostic Variable T_DIAG_MODBUS _ETH_SERVER_1 .*	Size	Description
%QX(n+2).6	tCommand. bDiag_22_reserved	BIT	Reserved.
%QX(n+2).7	tCommand. bDiag_23_reserved	BIT	Reserved.
%QB(n+3)	byDiag_3_reserved	BYTE	Reserved.
Communication statistics:			
%QW(n+4)	tStat. wActiveConnections	WORD	Number of established connections between client and server (0 to 64).
%QW(n+6)	tStat. wTimeoutClosedConnections	WORD	Connections counter, between the client and server, interrupted after a period of inactivity - time-out (0 to 65535).
%QW(n+8)	tStat. wClientClosedConnections	WORD	Connections counter interrupted due to customer request (0 to 65535).
%QW(n+10)	tStat. wRXFrames	WORD	Ethernet frames counter received by the server. An Ethernet frame can contain more than one request (0 to 65535).
%QW(n+12)	tStat. wRXRequests	WORD	Requests received by the server counter and answered normally (0 to 65535).
%QW(n+14)	tStat. wTXExceptionResponses	WORD	Requests received by the server counter and answered with exception codes (0 to 65535). The exception codes are listed below: 1: the function code (FC) is legal, but not supported. 2: relation not found in these data MODBUS. 3: illegal value for the address. 128: the master/client has no right to read or write. 129: MODBUS relation is disabled.
%QW(n+16)	tStat. wRXIllegalRequests	WORD	Illegal requests counter (0 to 65535).
%QW(n+18)	tStat. wDiag_18_Reserved	WORD	Reserved.

Table 123: MODBUS Server Diagnostics

Note:
Counters: all counters of the MODBUS Ethernet Server Diagnostics return to zero when the limit value 65535 is exceeded.

5.5.9.1.3. Mapping Configuration – Configuration via Symbolic Mapping

The MODBUS relations configuration, showed on figure below, follows the parameters described on table below:

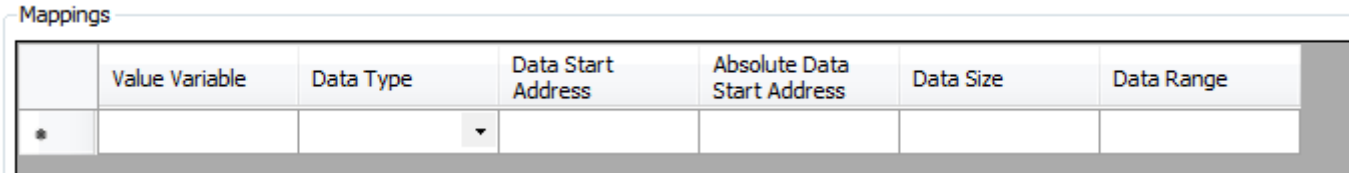


Figure 78: MODBUS Server Data Mappings Screen

Configuration	Description	Default Value	Options
Value Variable	Symbolic variable name	-	Name of a variable declared in a program or GVL
Data Type	MODBUS data type	-	Coil Input Status Holding Register Input Register
Data Start Address	Starting address of the MODBUS data	-	1 to 65536
Absolute Data Start Address	Start address of absolute data of Modbus as its type	-	-
Data Size	Size of the MODBUS data	-	1 to 65536
Data Range	Data range address configured	-	-

Table 124: MODBUS Ethernet Mappings Configuration

Notes:

Value Variable: this field is used to specify a symbolic variable in MODBUS relation.

Data Type: this field is used to specify the data type used in the MODBUS relation.

Data Start Address: data initial address of the MODBUS relation.

Absolute Data Start Address: absolute start address of the MODBUS data according to their type. For example, the Holding Register with address 5 has absolute address 400005. This field is read only and is available to assist in Client/Master MODBUS configuration that will communicate with this device. The values depend on the base address (offset) of each data type and allowed MODBUS address for each data type.

Data Size: the Data Size value sets the maximum amount of data that a MODBUS relation can access from the initial address. Thus, in order to read a continuous range of addresses, it is necessary that all addresses are declared in a single relation. This field varies according to the configured type of MODBUS data.

Data Range: is a read-only field and reports on the range of addresses that is being used by this mapping. It is formed by the sum of the fields *Data Start Address* and *Data Size*. There can be no range overlays with others mappings of the same *Data Type*.

ATTENTION

Unlike other tasks of an application, when a mark is reached at MainTask debugging, the MODBUS Ethernet Server instance task or any other MODBUS task will stop being executed at the moment it tries to write in the memory area. This occurs in order to maintain data consistency of memory areas while MainTask is not running.

5.5.9.2. MODBUS Server Ethernet Protocol Configuration via Direct Representation (%Q)

To configure this protocol using *Direct Representation (%Q)*, the user must perform the following steps:

- Configure the general parameters of MODBUS Server Protocol, such as: communication times, address and direct representation variables (%Q) to receive the diagnostics and control relation.

- Add and configure MODBUS relations, specifying the MODBUS data type, direct representation variables (%Q) to receive/write the data and amount of data to be reported.

The descriptions of each configuration are listed below in this section.

5.5.9.2.1. *General Parameters of MODBUS Server Protocol – Configuration via Direct Representation (%Q)*

The general parameters, found on the home screen of MODBUS protocol configuration (figure below), are defined as:

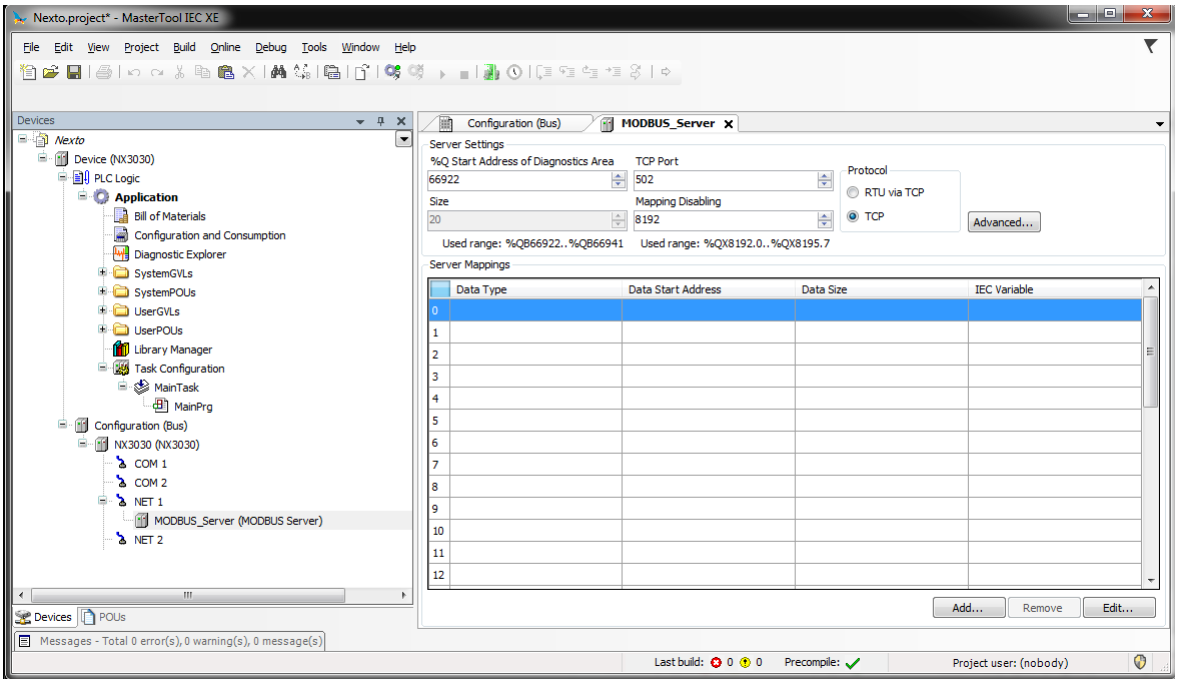


Figure 79: MODBUS Server Setup Screen

TCP port, protocol and direct representation variables (%Q) to control relations and diagnostics:

Configuration	Description	Default Value	Options
%Q Start Address of Diagnostics Area	Starting address of the diagnostic variables	-	0 to 2147483628
Size	Size of diagnostics	20	Disabled for editing
TCP Port	TCP Port	502	2 to 65534
Mapping Disabling	Starting address used to disable MODBUS relations	-	0 to 2147483644
Protocol	Protocol selection	TCP	RTU via TCP TCP

Table 125: Settings to control relations and diagnostics

Notes:

%Q Start Address of Diagnostics Area: this field is limited by the size of output variables addressable memory (%Q) at CPU, which can be found in section Memory.

TCP Port: if there are multiple instances of the protocol added in a single Ethernet interface, different TCP ports must be selected for each instance. Some TCP ports, among the possibilities mentioned above, are reserved and therefore cannot be used. See table Reserved TCP/UDP ports.

Mapping Disabling: composed of 32 bits, used to disable, individually, the 32 MODBUS relations configured in *Server Mappings* space. The relation is disabled when the corresponding bit is equal to 1, otherwise, the mapping is enabled. This field is limited by the size of output variables addressable memory (%Q) of each CPU, which can be found on Memory section.

Default Value: the factory default value cannot be set to the *%Q Start Address of Diagnostics Area* and *Mapping Disabling* fields, because the creation of a Protocol instance may be held at any time on application development. The MasterTool IEC XE software itself allocate a value, from the range of output variables of direct representation (%Q), not used yet.

The communication times of the MODBUS Server protocol, found on the *Advanced...* button of the configuration screen, are divided into: *Task Cycle (ms)* and *Connection Inactivity Time-out (s)*. Further details are described in MODBUS Server Protocol General Parameters – Configuration via Symbolic Mapping section.

The diagnostics and MODBUS commands are described in Table 123.

5.5.9.2.2. *Mapping Configuration – Configuration via Direct Representation (%Q)*

The MODBUS relations settings, viewed in the figures below, follow the parameters described in table below:

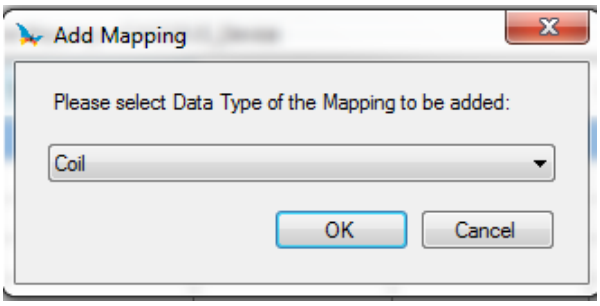


Figure 80: MODBUS Data Type

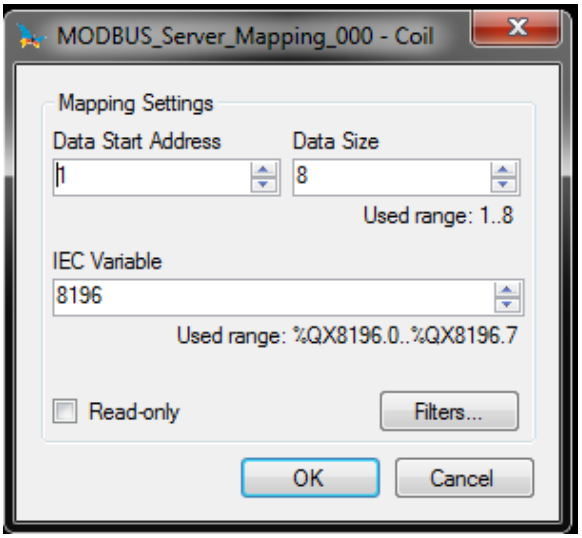


Figure 81: MODBUS Server Function

Configuration	Description	Default	Options
Data Type	MODBUS data type	Coil	Coil (1 bit) Holding Register (16 bits) Input Status (1 bit) Input Register (16 bits)
Data Start Address	MODBUS data initial address	1	1 to 65536
Data Size	MODBUS data quantity	8	1 to 65536 (Holding Register and Input Register) 8 to 65536 (Coil and Input Status)
IEC Variable	Variables initial address (%Q)	-	0 to 2147483647
Read-only	Allow reading only	Disabled	Enabled or Disabled

Table 126: Server Mappings

Notes:

Options: the values written in the column *Options* may vary according with the configured MODBUS data.

Data Size: the *Data Size* value sets the maximum amount of data that a MODBUS relation can access from the initial address. Thus, to read a continuous range of addresses, it is necessary that all addresses are declared in a single relation. This field varies according to the set MODBUS data type, that is, when selected *Coil* or *Input Status*, the field *Data Size* must be a number multiple of 8. It is also important to take care so the maximum value is not greater than the addressable output memory size and the attributed values aren't the same already used during the application.

ATTENTION

When accessing the communication data memory is between devices with different endianism (Little-Endian and Big-Endian), inversion of the read/write data may occur. In this case, the user must adjust the data in the application.

IEC Variable: in case the MODBUS data type is *Coil* or *Input Status* (bit), the IEC variables initial address will be in the format for example *%QX10.1*. However, if the MODBUS data type is *Holding Register* or *Input Register* (16 bits), the IEC variables initial address will be in the format *%QW*. This field is limited by the memory size of the addressable output variables (%Q) from each CPU, which can be seen on the Memory section.

Read-only: when enabled, it only allows the communication master to read the variable data. It does not allow the writing. This option is valid for the writing functions only.

Default: the default cannot be defined for the *IEC Variable* field as the creation of a protocol instance can be made at any moment within the application development, making the MasterTool IEC XE software allocate a value itself from the direct representation output variables range (%Q) still not used. The default cannot be defined for the *Data Size* field as it will vary according to selected MODBUS data type.

The settings present on the *Filters...* button, described in table below, are relative to the TCP communication filters:

Configuration	Description	Default Value	Options
Write Filter IP Address	Specifies a range of IPs with write access in the variables declared in the MODBUS interface.	0.0.0.0	0.0.0.0 to 255.255.255.255
Write Filter Mask	Specifies the subnet mask in conjunction with the IP filter parameter for writing.	0.0.0.0	0.0.0.0 to 255.255.255.255

Configuration	Description	Default Value	Options
Read Filter IP Address	Specifies a range of IPs with read access in the variables declared in the MODBUS interface.	0.0.0.0	0.0.0.0 to 255.255.255.255
Read Filter Mask	Specifies the subnet mask in conjunction with the IP filter parameter for reading.	0.0.0.0	0.0.0.0 to 255.255.255.255

Table 127: IP Filters

Note:

Filters: filters are used to establish a range of IP addresses that have write or read access to MODBUS relations, being individually configured. The permission criteria is accomplished through a logical AND operation between the Write Filter Mask and the IP address of the client. If the result is the same as the Write Filter IP Address, the client is entitled to write. For example, if the Write Filter IP Address = 192.168.15.0 and the Write Filter Mask = 255.255.255.0, then only customers with IP address = 192.168.15.x shall be entitled. The same procedure is applied in the Read Filter parameters to define the read rights.

In the previously defined relations, the maximum MODBUS data size can be 65536 (maximum value configured in the *Data Size* field). However, the request which arrives in the MODBUS Ethernet Server must address a subgroup of this mapping and this group must have, at most, the data size depending on the function code which is defined below:

- Read Coils (FC 1): 2000
- Read Input Status (FC 2): 2000
- Read Holding Registers (FC 3): 125
- Read Input Registers (FC 4): 125
- Write Single Coil (FC 5): 1
- Write Single Holding register (FC 6): 1
- Force Multiple Coils (FC 15): 1968
- Write Holding Registers (FC 16): 123
- Mask Write Register (FC 22): 1
- Read/Write Holding Registers (FC 23):
  - Read: 121
  - Write: 121

ATTENTION

Differently from other application tasks, when a depuration mark in the MainTask is reached, the task of an Ethernet MODBUS Server instance and any other MODBUS task will stop running at the moment that it tries to perform a writing in a memory area. It occurs in order to keep the consistency of the memory areas data while a MainTask is not running.

5.5.10. OPC DA Server

It's possible to communicate with the Nexto Series CPUs using the OPC DA (*Open Platform Communications Data Access*) technology. This open communication platform was developed to be the standard in industrial communications. Based on client/server architecture, it offers several advantages in project development and communication with automation systems.

A very common analogy to describe the OPC DA technology is of a printer. When correctly connected, the computer needs a driver to interface with the equipment. Similarly, the OPC helps with the interface between the supervision system and the field data on the PLC.

When it comes to project development, to configure the communication and exchange information between the systems is extremely simple using OPC DA technology. Using other drivers, based on addresses, it's necessary to create tables to relate tags from the supervision system with variables from the programmable controller. When the data areas are changed during the project, it's necessary to remap the variables and create new tables with the relations between the information on the PLC with the Supervisory Control And Data Acquisition system (SCADA).

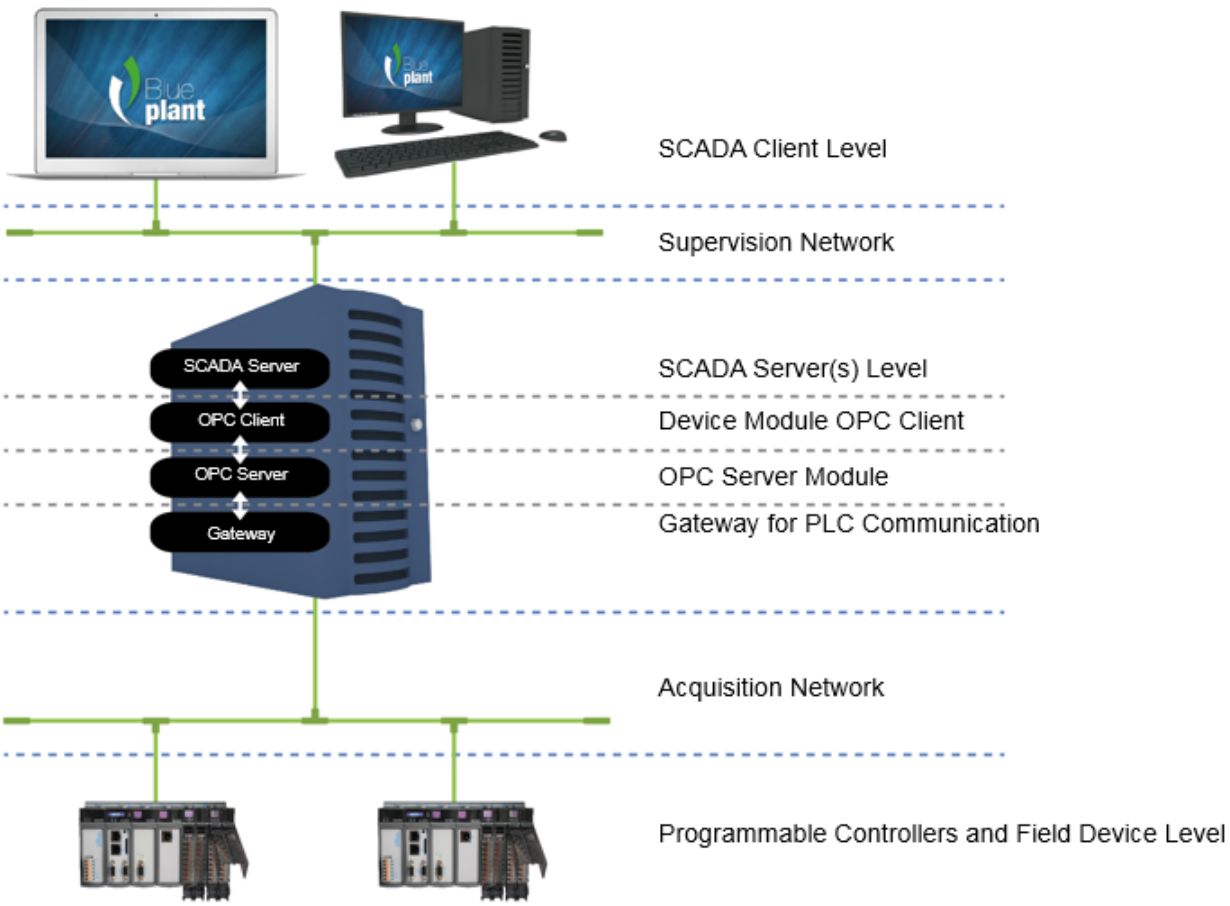


Figure 82: OPC DA Architecture

The figure above shows an architecture to communicate a SCADA system and PLCs in automation projects. All the roles present on a communication are explicit on this figure regardless of the equipment in which it's executed, since they can be done in the same equipment or in various ones. Each of the roles of this architecture are described on table below.

Role	Description
Programmable Controllers and Field Devices Level	The field devices and the PLCs are where the operation state and plant control information are stored. The SCADA system access the information on these devices and store on the SCADA server, so that the SCADA clients can consult it during the plant operation.
Acquisition Network	The acquisition network is where the requests for data collected by field devices travel, to request the data collected from the field devices.
Gateway for PLC Communication	A gateway enables the communication between the OPC DA Server and Nexto Series PLCs. A gateway in the same subnet of the PLC is always necessary, as described in chapter Communication Settings of MasterTool IEC XE User Manual – MU299609.
OPC Server Module	The OPC DA Server is a Module responsible of receiving the OPC DA requests and translate them to the communication with the field devices.

Role	Description
Device Module OPC Client	The OPC Client Device module is responsible for the requests to the OPC DA Server using the OPC DA protocol. The collected data is stored on the SCADA Server database.
SCADA Server Level	The SCADA Server is responsible for connecting to the various communication devices and store the data collected by them on a database, so that it can be consulted by the SCADA Clients.
Supervision Network	The supervision network is the network through which the SCADA Clients are connected to the SCADA Servers. In a topology in which there aren't multiple Client or where the Server and the Client are installed on the same equipment, this kind of network doesn't exist.
SCADA Client Level	The SCADA Clients are responsible for requesting to the SCADA Servers the necessary data to be shown in a screen where the operation of a plant is being executed. Through then it is possible to execute readings and writings on data stored on the SCADA Server database.

Table 128: Roles Description on an OPC DA Server Architecture

The relation between the tags on the supervision system and the process data on the controller variables is totally transparent. This means that, if there's an alteration on the data areas through the development of the project, it isn't necessary to rework the relations between the information on the PLC and the SCADA, just use the new variable provided by the PLC on the systems that request this data.

The use of OPC offers more productivity and connectivity with SCADA systems. It contributes with the reduction of applications development time and with the maintenance costs. It even makes possible the insertion of new data on the communication in a simplified form and with greater flexibility and interoperability between the automation system, due to the fact that it's an open standard.

The installation of the OPC DA Server is done altogether with MasterTool IEC XE installation, and its settings are done inside the tool. It's worth notice that the OPC is available only with the local Ethernet interface of the Nexto CPUs. The Ethernet expansion modules do not support this functionality.

5.5.10.1. Creating a Project for OPC DA Communication

Unlike the communication with drivers such as MODBUS and PROFIBUS DP, to set an OPC DA communication it's only necessary to correctly set the node and indicate which variables will be used in the communication. There are two ways to indicate which variables of the project will be available in the OPC DA Server. In both cases it's necessary to add the object *Symbol Configuration* to the application, in case it isn't present. To add it, right-click over the object *Application* and select the option.

ATTENTION

The variables shown in the objects *IoConfig_Globals*, *IoConfig_Application_Mappings* and *IoConfig_Global_Mappings* are used internally for I/O control and shouldn't be used by the user.

ATTENTION

In addition to the variables declared at SFC language POUs, some implicitly created variables are also shown. To each step created, a type *IecSfc.SFCStepType* variable is created, where the step states can be monitored, namely whether it is active or not and the time that it's active as in norm IEC 61131-1. To each transition, a BOOL type variable is created that defines if the transition is true or false. These variables are shown in the object *Symbol Configuration* that can be provided access to the OPC Client.

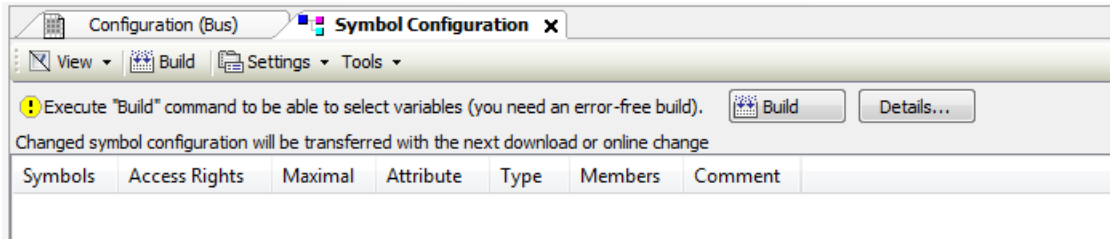


Figure 83: Symbol Configuration Object

The table below presents the descriptions of the *Symbol Configuration* object screen fields.

Field	Description
Symbols	Variable identifier that will be provided to the OPC DA Server.
Access Rights	Indicates what the possible access right level are in the declared symbol. When not utilized, this column remains empty, and the access right level is maximum. Otherwise the access right level can be modified by clicking over this field. The possible options are: Read only Write only Read and Write
Maximal	Indicates the maximum access right level that is possible to assign to the variable. The symbols hold the same meanings from the ones in Access Rights. It's not possible to change it and it's indicated by the presence or not of the *attribute 'symbol'*
Attribute	Indicates if *attribute 'symbol'* is being used when the variable is declared. When not used, this column remains empty. For the cases in which the attribute is used, the behavior is the following: attribute 'symbol' := 'read' the column shows attribute 'symbol' := 'write' the column shows attribute 'symbol' := 'readwrite' the column shows
Type	Data type of the declared variable.
Members	When the data type is a Struct, a button is enabled in this column. Clicking on the button will allow the selection of which elements of that struct will be provided to the OPC DA Server.
Comment	Variable comment, inserted on the POU or GVL where the variable was declared. To show up as a variable comment here, the comment must be entered one line before the variable on the editor, while in text mode, or in the comment column when in tabular mode.

Table 129: Symbol Configuration object screen fields description

When altering the project settings, such as adding or removing variables, it's necessary to run the command *Build*, in order to refresh the list of variables. This command must be executed until the message in Figure 83 disappear. After this, all available variables in the project, whether they are declared on POUs, GVLs or diagnostics, will be shown here and can be selected. The selected variables will be available on the OPC DA Server to be accessed by the Clients.

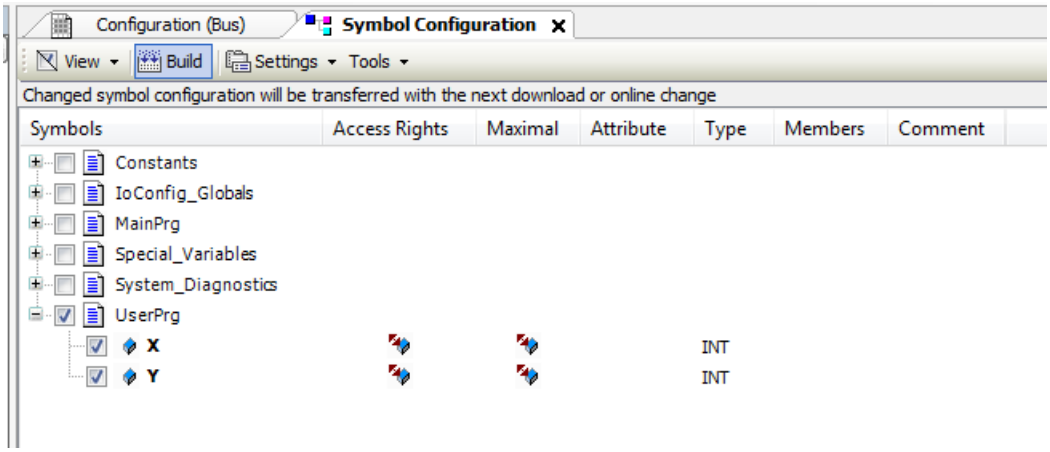


Figure 84: Selecting Variables on the Symbol Configuration

After this procedure, the project must be loaded into a PLC so the variables will be available for communication with the OPC DA Server. If the object Symbol Configuration screen is open and any of the variables, POUs or GVLs selected is changed, its name will appear with the red color. The situations in which this may happen is when a variable is deleted or the attribute value is modified.

It's also possible to set which variables will be available on the OPC DA Server through an attribute inserted directly on the POUs or GVLs where the variables are declared. When the *attribute 'symbol'* is present on the variable declaration, and it may be before the definition of the POU or GVL name, or to each variable individually, these variables are sent directly to the object *Symbol Configuration*, with a symbol in the *Attribute* column. In this case it's necessary, before loading the project into the CPU, to run the command *Build* from within the object *Symbol Configuration*.

The valid syntaxes to use the attribute are:

- *attribute 'symbol' := 'none'* – when the attribute value is *'none'*, the variables won't be available to the OPC DA Server and won't be shown in the object *Symbol Configuration* screen.
- *attribute 'symbol' := 'read'* - when the attribute value is *'read'*, the variables will be available to the OPC DA Server with read only access right.
- *attribute 'symbol' := 'write'* - when the attribute value is *'write'*, the variables will be available to the OPC DA Server with write only access right.
- *attribute 'symbol' := 'readwrite'* – when the attribute value is *'readwrite'*, the variables will be available to the OPC DA Server with read and write access right.

In the following example of variable declaration, the variables A and B settings allow that an OPC DA Server access them with read and write access. However the variable C cannot be accessed, while the variable D can be accessed with read only access rights.

```
{attribute 'symbol' := 'readwrite'}
PROGRAM UserPrg
VAR
A: INT;
B: INT;
{attribute 'symbol' := 'none'}
C: INT;
{attribute 'symbol' := 'read'}
D :INT;
END_VAR
```

When a variable with a type different from the basic types is defined, the use of the attribute must be done inside the declaration of this DUT and not only in the context in which the variable is created. For example, in the case of a DUT instance inside of a POU or a GVL that has an attribute, it will not impact in the behavior of this DUT instance elements. It will be necessary to apply the same access right level on the DUT declaration.

ATTENTION

The configurations of the symbols that will be provided to the OPC DA Server are stored inside the PLC project. By modifying these configurations it's necessary to load the application on the PLC so that it's possible to access those variables.

ATTENTION

When a variable is removed from the project and loaded on the PLC unchecking it from the object *Symbol Configuration*, the variable can no longer be read with the OPC Client. If the variable is added again to the project, with the same name and same context, and inserted on the object *Symbol Configuration*, it will be necessary to reboot the OPC Client to refresh the variable address reference, which will be created on a different memory area of the PLC.

5.5.10.2. Configuring a PLC on the OPC DA Server

The configuration of the PLC is done inside MasterTool IEC XE through the option available in the *Online*. It's necessary to run MasterTool IEC XE as administrator.

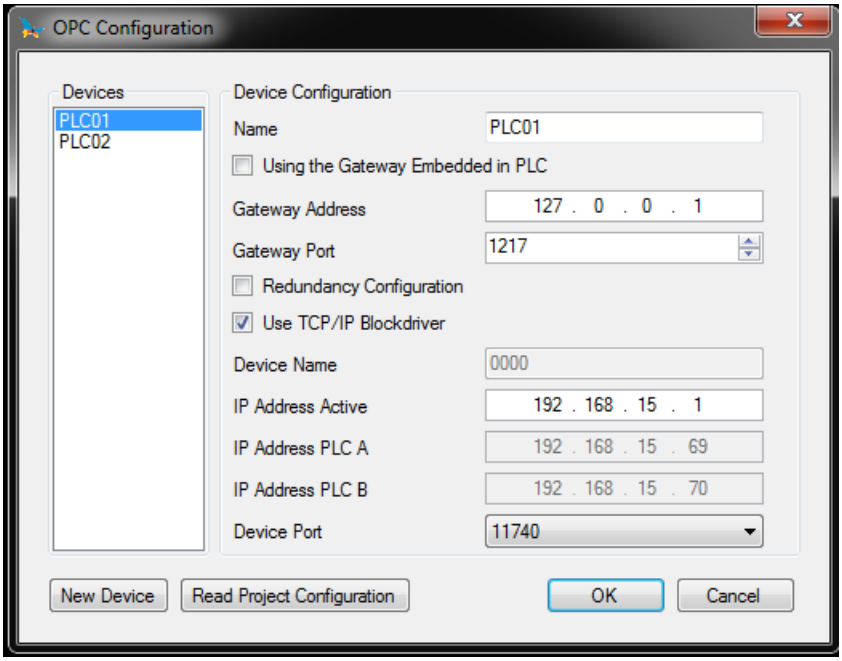


Figure 85: OPC DA Server Settings

The *Gateway Configuration* is the same set in the Gateway used for the communication between the MasterTool IEC XE and the PLC and described in Communication Settings, present in the MasterTool IEC XE User Manual – MU299609. If the configuration used is *localhost* the *Gateway Address* must be filled with 127.0.0.1. This configuration is necessary because the OPC DA Server uses the same communication gateway and the same protocol used for communication between PLC and MasterTool IEC XE.

There's the option *Using the Gateway Embedded in PLC* that can be selected when it's desired to use the Gateway that is in PLC itself. This option can be used to optimize the communication, since it prevent excess traffic through a particular station, when more than one station with OPC Client is connected to the same PLC.

To configure the PLC, there are two possible configuration types, depending on the selection of the checkbox *Use TCP/IP Blockdriver*. When the option isn't selected, the field *Device Name* must be filled with the name of the PLC. This is the name displayed by the PLC selected as active in the *Communication Settings* screen.

The other option is to use the *IP Address* of the Ethernet Interfaces. The same address set on the configuration screens must be put in this field. Furthermore, when this method is used, the port number must be set to 11740. The confirmation will save the OPC DA Server configurations.

Device Configuration	Description	Default Setting	Options
Name	PLC description inside the OPC DA Server configuration file. This field can have any name, but for organizational purposes, it's recommended to use the project name that is loaded in the PLC.	'PLC01'	This field is a STRING and it accepts alphanumeric (letters and numbers) characters and the "_" character. It's not allowed to initiate a STRING with numbers or with "_". It allows up to 49 characters.
Gateway Address	IP Address of the computer that the OPC DA Server is installed, for the cases in which all PLCs are in the same subnetwork. If there's some PLC that it's in another subnetwork, it must be specified the Gateway used in that subnetwork.	127.0.0.1	0.0.0.0 to 255.255.255.255
Gateway Port	TCP Port for the connection with the Gateway.	1217	2 to 65534
Device Name	It's the PLC name displayed in the *Communication Settings* of the *Device* tab. The name is the STRING before the hexadecimal value that is between *[ ]*. Enabled only when the checkbox *Use TCP/IP Blockdriver* is not selected.	'0000'	This field is a STRING and it accepts any characters, as done in the PLC name configuration in the Device *Communication Settings* tab. It allows up to 49 characters.
IP Address Active	IP address of the PLC. Enabled only when the checkbox *Use TCP/IP Blockdriver* is selected. It is used only when the setting is not redundant.	192.168.15.1	0.0.0.0 to 255.255.255.255
IP Address PLC A	IP address of the PLC A. Enabled only when the configuration is redundant. It is the primary PLC address to which the server will communicate if there is no failure.	192.168.15.69	0.0.0.0 to 255.255.255.255
IP Address PLC B	IP address of the PLC B. Enabled only when the configuration is redundant. It is the secondary PLC address to which the server will communicate if a failure occurs.	192.168.15.70	0.0.0.0 to 255.255.255.255
Device Port	TCP Port. Enabled only when the checkbox *Use TCP/IP Blockdriver* is selected.	11740	11740 or 11739

Table 130: Configuration Parameter of each PLC for the OPC DA Server

When a new PLC needs to be configured on the OPC DA Server, simply press the *New Device* button and the configuration will be created. When the setup screen is accessed, a list of all PLCs already configured on the OPC DA Server will be displayed. Existing configurations can be edited by selecting the PLC in the *Devices* list and editing the parameters. The PLCs settings that are no longer in use can be deleted. The maximum number of PLCs configured in an OPC DA Server is 16.

If the automation architecture used specifies that the OPC DA Server must be ran on a computer that does not execute communication with the PLC via MasterTool IEC XE, the tool must be installed on this computer to allow OPC DA Server configuration in the same way as done in other situations.

ATTENTION

To store the OPC DA Server configuration, the MasterTool IEC XE must be run with administrator rights on the Operational System. Depending on the OS version, this privilege must be done in the moment that the program is executed: right-click the MasterTool IEC XE icon and choose *Run as Administrator*.

ATTENTION

The settings of a PLC on the OPC DA Server are not stored in the project created in MasterTool IEC XE. For this reason, it can be performed with an open or closed project. The settings are stored in a configuration file where the OPC DA Server is installed. When changing the settings, it is not required to load the application on the PLC, but depending on the OPC Client it may be necessary to reconnect to the server or load the settings for the data to be updated correctly.

5.5.10.2.1. Importing a Project Configuration

Using the button *Read Project Configuration*, as shown in Figure 85, you can assign the configuration of the open project to the PLC configuration that is being edited. For this option to work correctly, there must be an open project and an *Active Path* should be set as described in *Communication Settings*, present in the MasterTool IEC XE User Manual – MU299609. If any of these conditions is not met an error message will be displayed and no data will be modified.

When the above conditions are valid, the PLC settings receive the parameters of the opened project. The *IP Address* and *Gateway Port* information are configured as described in *Communication Settings* according to the *Active Path*. However, the *IP Address* settings are read from NET 1 Ethernet interface settings. The port for connection to the PLC is always assigned in this case as 11740.

5.5.10.3. Configuration with the PLC on the OPC DA Server with Connection Redundancy

It's possible to configure the OPC DA Server for it to operate with connection redundancy. This way, the OPC DA Server will communicate preferably with one PLC, but when, by any reason, it can't establish communication with this PLC, a second PLC, also configured, will be accessed. This configuration is especially important for the communication between SCADA systems and the Nexto Series PLCs with Half-Cluster redundancy, where there's a PLC in active state executing the process, and another PLC in stand-by state, ready to take control of the process if some kind of failure occurs.

The project setup in these cases is similar to what is described in Creating a Project for OPC DA Communication. However, when a Project is created with Redundant Half-Cluster and the communication with the supervisory system will be through the OPC DA Server, it's necessary to select the *Configuration of OPC DA communication* option as enabled during the MasterTool IEC XE Project Creation Wizard. By enabling this option, the project will create the code needed to run the communication with OPC connection redundancy.

In the redundant case, a variable is declared within the POU named NonSkippedPrg. This POU is executed in both PLCs, regardless of redundancy state. Within this POU, a BOOL type variable is created, used to control the connection with the OPC DA Server named *OPCRedundancyActive*. This variable can be accessed from any application point through the whole context, i.e. *Application.NonSkippedPrg.OPCRedundancyActive*. It is declared in the *Symbol Configuration* object with the right read only by the SCADA. When the value of the variable is TRUE, data is read by connecting with this PLC. This way, every time there is a status change among PLCs, the variable state will also change, remaining in the state TRUE in the PLC which is in the redundancy active state.

The NonSkippedPrg program code, in ST language, is as follow:

```
PROGRAM NonSkippedPrg
VAR
  {attribute 'symbol' := 'read'}
  OPCRedundancyActive : BOOL;
END_VAR

IF fbRedundancyManagement.m_fbDiagnosticsLocal.eRedState = REDUNDANCY_STATE.
   ACTIVE THEN
  OPCRedundancyActive := TRUE;
ELSE
  OPCRedundancyActive := FALSE;
END_IF
```

The *NonSkippedPrg* program code can be edited as long as the user watch out not to change the above code. This code tests the state of redundancy and writes a BOOL type variable called *OPCRedundancyActive* with it. If the PLC is the active, the variable value is TRUE, otherwise it's FALSE. This variable receives the attribute *attribute 'symbol' := 'read'* to allow the OPC DA Server to access the content and define where the information should be read.

If it's decided to add OPC communication after the creation of the project, it is possible to configure the OPC by adding the above code in the *NonSkippedPrg* program and adding the *Symbol Configuration* object to the project.

For the configuration of the redundant PLC on the OPC DA Server, it's necessary to enable the *Redundancy Configuration* option in the configuration screen as shown in Figure 85. When this option is selected, the option *Use TCP/IP Blockdriver* will always be used. In addition, the *IP Address PLC A* and *IP Address PLC B* fields will be enabled as described in Table 130. These *IP Addresses* are configured in the same Ethernet interfaces within the PLC project with Half-Cluster redundancy. For ease of configuration when a redundant project is open, the *Read Project Configuration* button can be used.

ATTENTION

The OPC DA Server connection redundancy is done through only one Server. For the cases in which a better data availability for the supervision systems is desired, a redundant SCADA Server architecture must be adopted. In this cases it isn't required any OPC DA Server configuration. Refer to the SCADA system documentations to see which configurations are needed for the operation of the redundant architecture.

5.5.10.4. OPC DA Communication Status and Quality Variables

For each PLC created in the OPC DA Server, status variables are generated, named *_CommState* and *_CommStateOK*. The *_CommState* variable indicates the communication between the OPC and the PLC state. This state can interpreted by the OPC Clients according to table below.

State	Value	Description
STATE_TERMINATE	-1	If the communication between the OPC DA Server and the OPC Client is terminated, this value will be returned. When there's more than one OPC Client simultaneously connected, this return will occur on the disconnection of the latter connected one. Besides the fact that this state is in the variable, it's value can't be visualized because it only changes when there's no longer a connection with the client.
STATE_PLC_NOT_CONNECTED	0	The PLC configured in the OPC DA Server is not connected. It can happen if the configuration is incorrect (wrong PLC and/or Gateway IP Address) or the PLC is unavailable in that moment.

State	Value	Description
STATE_PLC_CONNECTED	1	The PLC configured in the OPC DA Server is connected. This is a transitory state during the connection.
STATE_NO_SYMBOLS	2	There are no symbols (variables) available in the PLC configured in the OPC DA Server. It can happen when there are no symbols or there isn't a project loaded on the PLC.
STATE_SYMBOLS_LOADED	3	Finished the process of reading the symbols (variables) from the PLC configured in the OPC DA Server. This is a transitory state during the connection.
STATE_RUNNING	4	After the reading of the symbols (variables) the OPC DA Server is running the periodic update of the values of the available symbols in each configured PLC.
STATE_DISCONNECT	5	There has been a disconnection with the PLC configured in the OPC DA Server.
STATE_NO_CONFIGURATION	6	When the OPC configuration (stored in an OPCServer.ini file) has a wrong syntax, the variable value will be this. Generally, this behavior is not observed for the MasterTool IEC XE maintains this configuration valid.

Table 131: Description of the Communication states between OPC DA Server and the PLC

The *_CommStateOK* is a variable of the Bool type that indicates if the communication between the OPC DA Server and the PLC is working. When the value is TRUE, it indicates that the communication is working correctly. If the value is FALSE, for some reason it isn't possible to communicate with the PLC.

In addition to monitoring the communication status, the OPC Client can access information on the quality of communication. The quality bits form a byte. They are divided into three groups of bits: *Quality*, *Substatus* and *Limit*. The bits are distributed as follows *QQSSSSLL*, in which *QQ* are the *Quality* bits, *SSSS Substatus* bits and *LL Limit* bits. In this case the *QQ* bits are the most significant in the byte, while the *LL* bits are the least significant.

QQ	Bits values	Definition	Description
0	00SSSSLL	Bad	The value read can't be used because there's some problem with the connection. It's possible to monitor the value of *_CommState* and diagnose the problem.
1	01SSSSLL	Uncertain	The quality can't be defined and may be presented in the Substatus field.
2	10SSSSLL	NA	This value is reserved and isn't used by the OPC standard.
3	11SSSSLL	Good	The quality is good and the value read can be used.

Table 132: Description of the OPC Quality value

Table 132 presents the possible quality values. The OPC DA Server only returns *Good* and *Bad* Quality values. A OPC Client can maintain the quality as *Uncertain* due to failures in which it can't establish a connection to the Server. In case of monitoring of the 8 quality bits directly from the OPC DA Server, the *Substatus* and *Limit* fields shall be null and the *Good* Quality will be presented as the value 192 and the *Bad* Quality will be value 0.

5.5.10.5. Limits of Communication with OPC DA Server

The table below presents the OPC DA Server configuration limits.

Maximum number of variables communicating with a single PLC	-
Maximum number of PLCs in an OPC DA Server	16
Maximum number of simultaneous connections of an OPC DA Server in a single PLC	8

Table 133: OPC DA Server Communication Limits

Note:

Maximum number of variables communicating with a single PLC: There is no configuration limit. The maximum possible number of variables depends on the processing capacity of the device.

ATTENTION

The Maximum number of simultaneous connections of an OPC DA Server in a single PLC is shared with connections made with the MasterTool IEC XE. I.e. the sum of connections of OPC DA Server and MasterTool IEC XE should not exceed the maximum quantity defined in Table 133.

The communication between the OPC DA Server and the PLC uses the same protocol used in the MasterTool IEC XE communication with the PLC. This protocol is only available for the Ethernet interfaces of the Nexto Series CPUs, it's not possible to establish this kind of communication with the Ethernet expansion modules.

When a communication between the OPC DA Server and the PLC is established, these two elements start a series of transactions aimed at solving the addresses of each declared variables, optimizing the communication in data reading regime. Besides, it's also resolved in this stage the communication groups used by some Clients in order to optimize the communication. This initial process demands some time and depends on the quantity of mapped variables and the processing capacity of the device.

5.5.10.6. Accessing Data Through an OPC DA Client

After the configuration of the OPC DA Server, the available data on all PLCs can be accessed via an OPC Client. In the configuration of the OPC Client, the name of the OPC DA Server must be selected. In this case the name is *CoDeSys.OPC.DA*. The figure below shows the server selection on the client driver of the BluePlant SCADA software.

ATTENTION

The same way that in MasterTool IEC XE, some tools must be executed with administrator privileges in the Operational System for the correct functioning of the OPC Client. Depending on the OS version, this privilege must be activated in the moment that the program is executed. To do this, right-click MasterTool IEC XE icon and choose *Run as Administrator*.

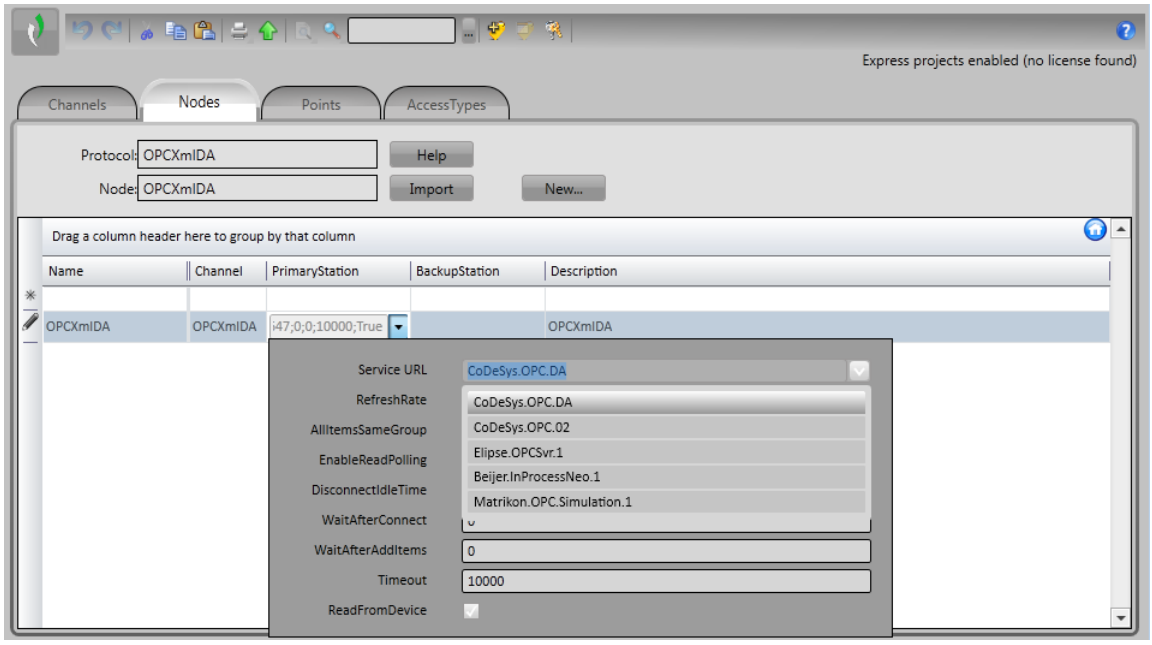


Figure 86: Selecting the OPC DA Server in the Client Configuration

In cases where the server is remotely located, it may be necessary to add the network path or IP address of the computer in which the server is installed. In these cases, there are two configuration options. The first is to directly configure it, being necessary to enable the COM/DCOM Windows Service. However, a simpler way is to use a tunneller tool that abstracts the COM/DCOM settings, and enable a more secure communication between the Client and the Server. For more information on this type of tool, refer to the *NAP151 - Tunneller OPC*.

Once the Client connects with the Server, it's possible to use the TAGs import commands. These commands consult the information declared in the PLC, returning a list with all the symbols available in it.

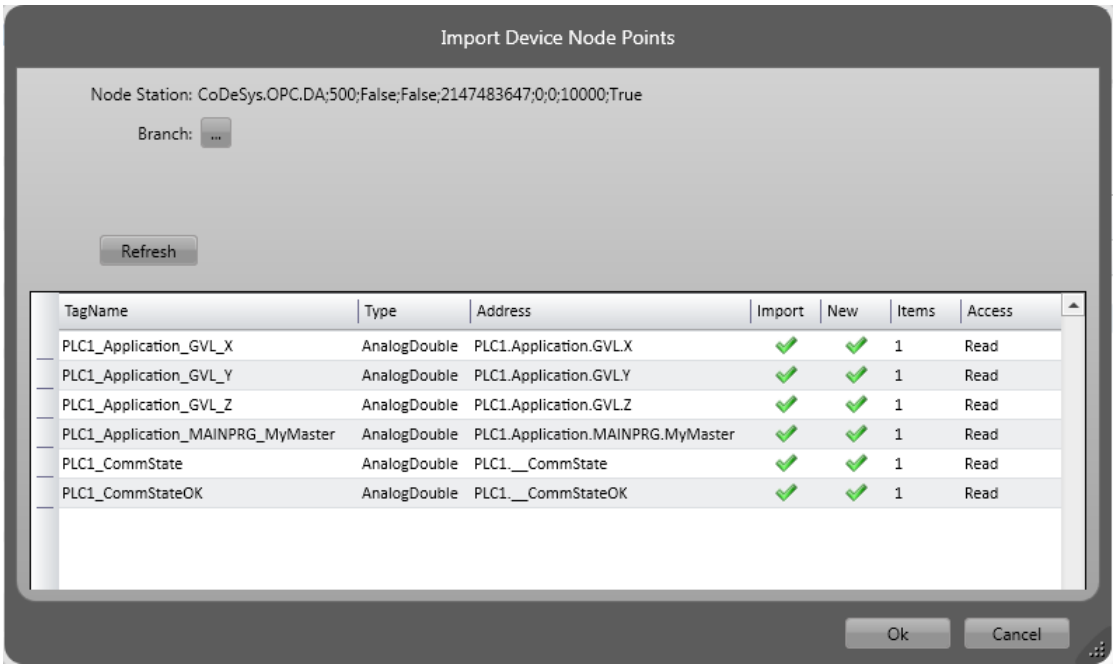


Figure 87: Symbols list consulted by the OPC Client

The list of selected variables will be included in the Client communication list and can be used, for example, in a SCADA system screen.

ATTENTION

The simulation mode of MasterTool IEC XE software can be used for OPC communication tests. The information on how to configure it are presented in the *Testing an OPC Communication using the Simulator* section of the MasterTool IEC XE User Manual – MU299609.

5.5.11. OPC UA Server

The OPC UA protocol is an evolution of the OPC family. Independent of platform, it is designed to be the new standard used in industrial communications.

Based on the client/server architecture, the OPC UA protocol offers numerous advantages in the development of design and facilities in communication with the automation systems.

When it comes to project development, configuring communication and exchanging information between systems is extremely simple using OPC UA technology. Using other address-based drivers, it is necessary to create tables to relate the supervision system tags and programmable controller variables. When data areas change during project development, it is necessary to redo the mappings and new tables with the relationships between the PLC information and the SCADA system.

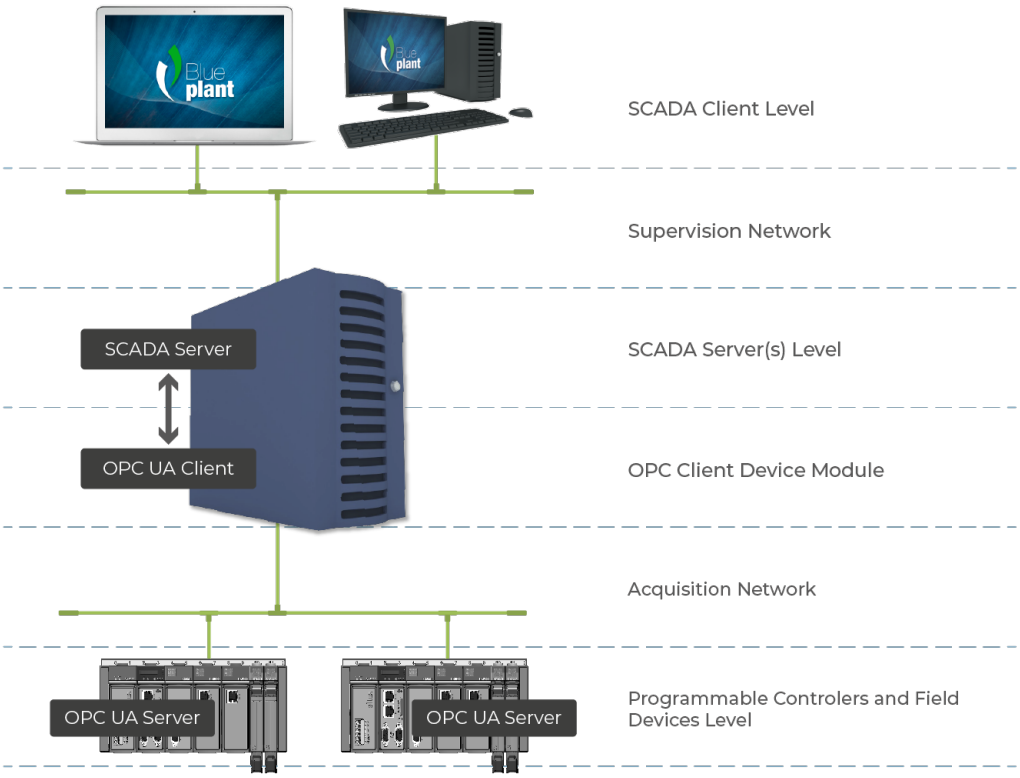


Figure 88: OPC UA Architecture

The figure above presents a typical architecture for SCADA system communication and PLCs in automation design. All roles present in the communication are explicit in this figure regardless of where they are running, they may be on the same equipment or on different equipment. Each of the roles of this architecture is described in table below.

Role	Description
Programmable Controllers and Field Devices Level	The field devices and the PLCs are where the operation state and plant control information are stored. The SCADA system access the information on these devices and store on the SCADA server, so that the SCADA clients can consult it during the plant operation.
OPC UA Server Modules	The OPC UA Server is an internal module of the PLCs responsible for receiving the OPC UA requests and translating them for communication with the field devices.
Acquisition Network	The acquisition network is the network in which OPC UA messages travel to request the data that is collected from the PLCs and field devices.
OPC Client Device Module	The OPC UA Client module, which is part of the SCADA Server, is responsible for making requests to the OPC UA Servers using the OPC UA protocol. The data collected by it is stored in the SCADA Server database.
SCADA Server Level	The SCADA Server is responsible for connecting to the various communication devices and store the data collected by them on a database, so that it can be consulted by the SCADA Clients.
Supervision Network	The supervisory network is the network by which SCADA Clients are connected to SCADA Servers, often using a proprietary SCADA system protocol. In a topology in which multiple Clients are not used or the Server and Client are installed in the same equipment, there is no such network, and in this case this equipment must directly use the OPC UA protocol for communication with the PLC.
SCADA Client Level	The SCADA Clients are responsible for requesting to the SCADA Servers the necessary data to be shown in a screen where the operation of a plant is being executed. Through then it is possible to execute readings and writings on data stored on the SCADA Server database.

Table 134: Roles Description on an OPC UA Server Architecture

When using the OPC UA protocol, the relationship between the tags of the supervisory systems and the process data in the controller variables is completely transparent. This means that if data areas change during project development, there is no need to re-establish relationships between PLC information and SCADA. Simply use the new variable provided by the PLC in the systems that request this data.

The use of OPC UA offers greater productivity and connectivity with SCADA systems. It contributes to reduced application development time and maintenance costs. It also enables the insertion of new data in the communication in a simplified way with greater flexibility and interoperability among the automation systems as it is an open standard.

It is worth noting that the OPC UA is only available on the local Ethernet interfaces of the Nexto CPUs. Ethernet expansion modules do not support this functionality.

5.5.11.1. Creating a Project for OPC UA Communication

The steps for creating a project with OPC UA are very similar to the steps described in the section Creating a Project for OPC DA Communication. As with the OPC DA protocol, the configuration of the OPC UA protocol is based on the configuration of the *Symbol Configuration*. To enable the OPC UA, simply enable the *Support OPC UA Features* option in the configuration, as shown in figure below.

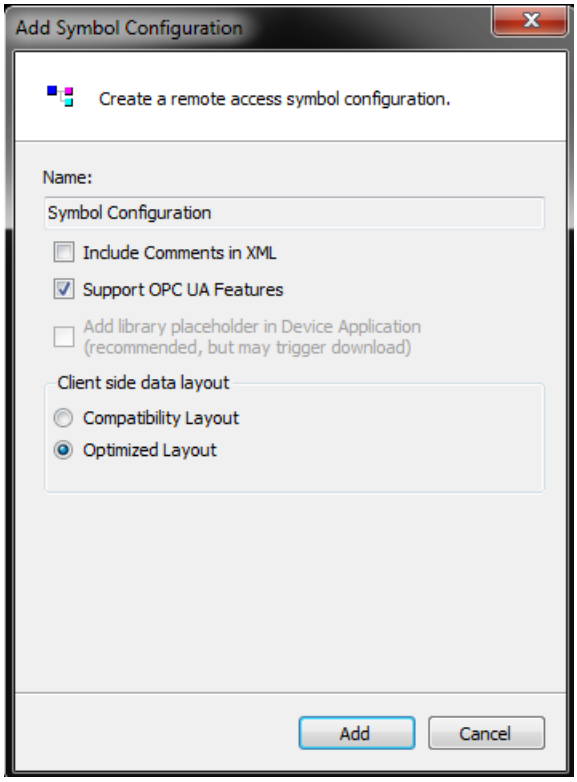


Figure 89: Symbol Configuration Object

ATTENTION

When enabling OPC UA protocol support, OPC DA protocol support is still enabled. You can enable OPC UA and OPC DA communications at the same time to report the variables configured on the *Symbol Configuration* object or via attributes.

Another way to access this configuration, once already created a project with the *Symbol Configuration* object, is given by accessing the *Settings* menu of the configuration tab of the *Symbol Configuration*. Simply select the option *Support OPC UA features* to enable support for the OPC UA protocol, as shown in figure below.

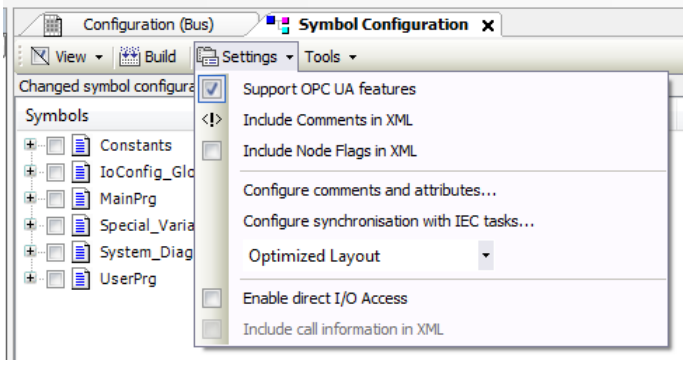


Figure 90: Enabling OPC UA in Object Symbol Configuration

After this procedure the project can be loaded into a PLC and the selected variables will be available for communication with the OPC UA Server.

5.5.11.2. Types of Supported Variables

This section defines the types of variables that support communication via the OPC UA protocol, when declared within GVLs or POUs and selected in the *Symbol Configuration* object (see previous section).

The following types of simple variables are supported:

- BOOL
- SINT
- USINT / BYTE
- INT
- UINT / WORD
- DINT
- UDINT / DWORD
- LINT
- ULINT / LWORD
- REAL
- LREAL
- STRING
- TIME
- LTIME

You can also use structured types (STRUCTs or Function Blocks) created from previous simple types.

Finally, it is also possible to create arrays of simple types or of structured types.

5.5.11.3. Limit Connected Clients on the OPC UA Server

The maximum number of OPC UA clients connected simultaneously in a PLC is 8 (eight).

5.5.11.4. Limit of Communication Variables on the OPC UA Server

There is no configuration limit. The maximum possible number of variables depends on the processing capacity of the device.

When a communication is established between the OPC UA Server and the PLC, these two elements initiate a series of transactions that aim to solve the address of each declared variable, optimizing the communication in regime of reading of data. In addition, at this stage, the classifications of the communication groups used by some Clients are also resolved in order to optimize communication. This initial process takes some time and depends on the amount of variables mapped and the processing capacity of the device.

5.5.11.5. Encryption Settings

If desired, the user can configure encryption for OPC UA communication using the *Basic256SHA256* profile, for a secure connection (cyber security).

To configure encryption on an OPC UA server, you must create a certificate for it using the following steps in the MasterTool programmer:

1. Define an active path for communication with the controller (no login required);
2. From the *View* menu, select *Security Screen*;
3. Click the *Devices* tab on the left side of this screen;
4. Click the icon  to perform a refresh;
5. Click on the *Device* icon, below which will open several certificates (*Own Certificates, Trusted Certificates, Untrusted Certificates, Quarantined Certificates*);
6. Click the icon  to generate a certificate and select the following parameters:
   - *Key length* (bit): 3072
   - *Validity period* (days): 365 (can be modified if desired)
7. Wait while the certificate is calculated and transferred to the controller (this may take a few minutes);
8. Reboot the controller.
9. On the OPC UA client, perform the necessary procedures to connect to the OPC UA server and generate a certificate with the *Basic256Sha256* profile (see specific OPC UA client manual for details);

10. Back to MasterTool, click on the icon of the *Security Screen* to perform a refresh;
11. On the *Security Screen*, select the "*Quarantined Certificates*" folder under the *Device*. In the right panel you should observe a certificate requested by the OPC UA client;
12. Drag this certificate to the folder "*Trusted Certificates*";
13. Proceed with the settings in the OPC UA client (see specific OPC UA client manual for details).

To remove encryption previously configured on a controller, you must do the following:

1. Define an active path for communication with the controller (no login required);
2. From menu *View*, select *Security Screen*;
3. Click on the *Devices* on the left side of this screen;
4. Click the icon to perform a refresh;
5. Click on the *Device* icon, below which will open several certificates (*Own Certificates, Trusted Certificates, Untrusted Certificates, Quarantined Certificates*);
6. Click the folder "*Own Certificates*" and in the right panel select the certificate (OPC UA Server);
7. Click the icon to remove this project and driver certificate;
8. Reset (turn off and on) the controller.

5.5.11.6. Main Communication Parameters Adjusted in an OPC UA Client

Some OPC UA communication parameters are configured on the OPC UA client, and negotiated with the OPC UA server at the time the connection between both is established. The following subsections describe the main OPC UA communication parameters, their meaning, and care to select appropriate values for them.

In an OPC UA client it is possible to group the variables of a server into different *subscriptions*. Each *subscription* is a set of variables that are reported in a single communication packet (*PublishResponse*) sent from the server to the client. The selection of the variables that will compose each subscription is made in the OPC UA client.

ATTENTION

Grouping variables into multiple *subscriptions* is interesting for optimizing the processing capacity and consumption of Ethernet communication bandwidth. Such aspects of optimization are analyzed in greater depth in the OPC UA Server user manual MU214609, where some rules for the composition of *subscriptions* are suggested. This user manual also discusses in more depth several concepts about the OPC UA protocol.

Some of the communication parameters described below must be defined for the server as a whole, others for each *subscription*, and others for each variable that makes up a *subscription*.

5.5.11.6.1. Endpoint URL

This parameter defines the IP address and TCP port of the server, for example:

opc.tcp://192.168.17.2:4840

In this example, the IP address of the controller is 192.168.17.2.

The TCP port should always be 4840.

5.5.11.6.2. Publishing Interval (ms) e Sampling Interval (ms)

The *Publishing Interval* parameter (unit: milliseconds) must be set for each *subscription*.

The *Sampling Interval* parameter must be set for each variable (unit: milliseconds). However, in many OPC UA clients, the *Sampling Interval* parameter can be defined for a *subscription*, being the same for all the variables grouped in the *subscription*.

Only the variables of a *subscription* whose values have been modified are reported to the client through a *Publish Response* communication packet. The *Publishing Interval* parameter defines the minimum interval between consecutive *Publish Response* packets of the same *subscription*, in order to limit the consumption of processing and Ethernet communication bandwidth.

To find out which subscription variables have changed and are to be reported to the client in the next *Publish Response* packet, the server must perform comparisons, and such (*samplings*) are performed by the same with the *Sampling Interval*. It is recommended that the value of *Sampling Interval* varies between 50% and 100% of the value of the *Publishing Interval*, because there is a relatively high processing consumption associated with the comparison process executed in each *Sampling Interval*.

It can be said that the sum between *Publishing Interval* and *Sampling Interval* is the maximum delay between changing a value on the server and the transmission of the *Publish Response* packet that reports this change. Half of this sum is the average delay between changing a value on the server and the transmission of the *Publish Response* packet that reports this change.

5.5.11.6.3. Lifetime Count e Keep-Alive Count

These two parameters must be configured for each *subscription*.

The purpose of these two parameters is to create a mechanism for deactivating a *subscription* on the initiative of the server, in case it does not receive customer's *PublishRequest* communication packets for this *subscription* for a long time. *PublishRequest* packets must be received by the server so that it can broadcast *Publish Response* packets containing the subscription variables that have changed their values.

If the server does not receive *PublishRequest* packets for a time greater than *Lifetime Count* multiplied by *Publishing Interval*, the server deactivates the *subscription*, which must be re-created by the client in the future if desired.

In situations where the variables of a *subscription* do not change, it could be a long time without the transmission of *PublishResponses* and consequently *PublishRequests* that succeed, causing an undesired deactivation of the *subscription*. To prevent this from happening, the *Keep-Alive Count* parameter was created. If there are no *subscription* data changes for a time equal to *Keep-Alive Count* multiplied by *Publishing Interval*, the server will send a small empty *Publish Response* packet indicating that no variable has changed. This empty *Publish Response* will authorize the client to immediately send the next *PublishRequest*.

The *Keep-Alive Count* value must be less than the *Lifetime Count* value to prevent unwanted deactivation of the *subscription*. It is suggested that *LifeTime Count* be at least 3 times larger than *Keep-Alive Count*.

5.5.11.6.4. Queue Size e Discard Oldest

These parameters must be maintained with the following fixed values, which are usually the default values on the clients:

- Queue Size: 1
- Discard Oldest: enable

According to the OPC UA standard, it is possible to define these parameters for each variable. However, many clients allow you to define common values for all variables configured in a *subscription*.

Queue Size must be retained with value 1 because there is no event support in this implementation of the OPC UA server, so it is unnecessary to define a queue. Increasing the value of *Queue Size* may imply increase communication bandwidth and CPU processing, and this should be avoided.

Discard Oldest must be maintained with the *enable* value, so that the *Publish Response* package always reports the most recent change of value detected for each variable.

5.5.11.6.5. Filter Type e Deadband Type

These parameters must be maintained with the following fixed values, which are usually the default values in the clients:

- *Filter Type: DataChangeFilter*
- *Deadband Type: none*

According to the OPC UA standard, it is possible to define these parameters for each variable. However, many clients allow you to define common values for all variables configured in a *subscription*.

The *Filter Type* parameter must be of *DataChangeFilter*, indicating that value changes in the variables should cause it to be transmitted in a *Publish Response* package.

Deadband Type should be kept in *"none"* because there is no implementation of *deadbands* for analog variables. In this way, any change of the analog variable, however minimal, causes its transmission in a *Publish Response* package.

To reduce processing power and Ethernet communication bandwidth, you can deploy *deadbands* on your own as follows:

- Do not include the analog variable in a *subscription*;
- Instead, include in a *subscription* an auxiliary variable linked to the analog variable;
- Copy the analog variable to the auxiliary variable only when the user-managed *deadband* is extrapolated.

5.5.11.6.6. PublishingEnabled, MaxNotificationsPerPublish e Priority

It is suggested that the following parameters be maintained with the following values, which are usually the default values in the clients:

- *PublishingEnabled: true*
- *MaxNotificationsPerPublish: 0*
- *Priority: 0*

These parameters must be configured for each *subscription*.

PublishingEnable must be *"true"* so that the *subscription* variables are reported in case of change of value.

MaxNotificationsPerPublish indicates how many of the variables that have changed value can be included in the same *Publish Response* package. The special value *"0"* indicates that there is no limit to this, and it is recommended to use this value so that all changed variables are reported in the same *Publish Response* package.

Priority indicates the relative priority of this *subscription* over others. If at any given moment the server should send multiple *Publish Response* packages of different *subscriptions*, it will prioritize the one with the highest value of priority. If all *subscriptions* have the same priority, *Publish Response* packets will be transmitted in a fixed sequence.

5.5.11.7. Accessing Data Through an OPC UA Client

After configuration of the OPC UA Server the data available in all PLCs can be accessed via a Client OPC UA. In the configuration of the OPC UA Client, the address of the correct OPC UA Server must be selected. In this case the address *opc.tcp://ip-address-of-device:4840*. The figure below shows the server selection in the SCADA BluePlant client software driver.

ATTENTION

Like MasterTool IEC XE, some tools need to be run with administrator rights on the Operating System for the correct operation of the OPC UA Client. Depending on the version of the Operating System this right must be authorized when running the program. For this operation right click on the tool executable and choose the option *Run as administrator*.

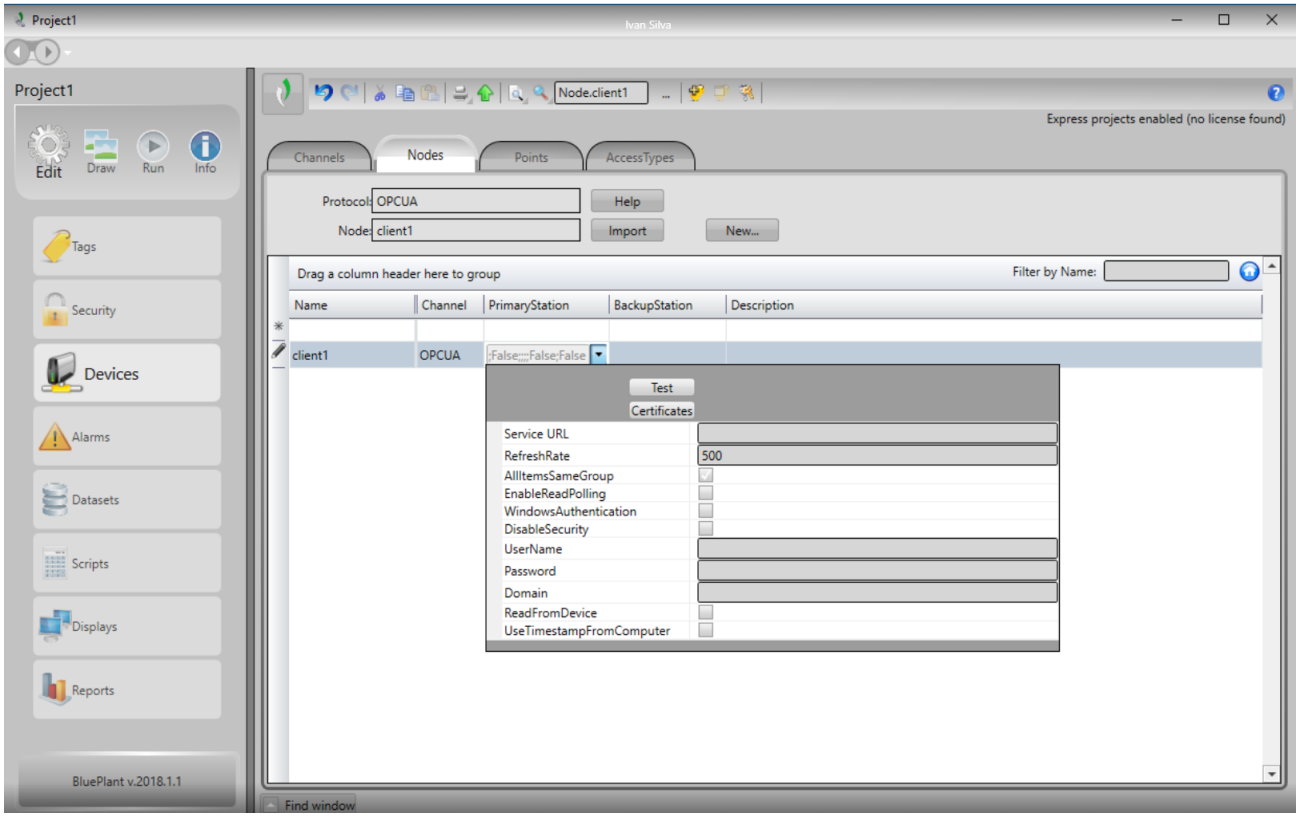


Figure 91: Selecting OPC UA Server in Client Configuration

Once the Client connects to the Server, TAG import commands can be used. These commands query information declared in the PLC, returning a list with all the symbols made available by the PLC.

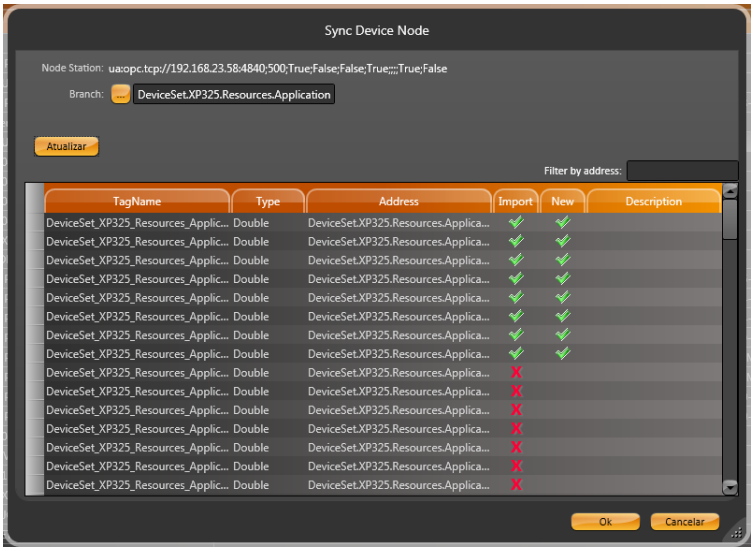


Figure 92: List of Symbols Browsed by OPC UA

The list of selected variables will be included in the Client's communications list and can be used, for example, in screens of a SCADA system.

5.5.12. EtherCAT Master

EtherCAT (*Ethernet Control Automation Technology*) is a master-slave architecture protocol with high performance, for deterministic Ethernet, that allows real time performance as it updates 1000 distributed I/O in 30 μS or 100 servomotors axis each 100 μS using twisted pair cables or optic fiber. Besides, it supports flexible topology, allowing for line, tree and/or star connections.

An Ethernet frame can be processed in real time instead of being received, interpreted and copied as process data in each connection. The FMMU (*Fieldbus Memory Management Unit*) in each Slave node reads the data that are addressed to it at the same time that the telegram is forwarded to the next device. In a similar way, the input data are inserted as the telegram is passed. Because of this, the frames are delayed just a few nanoseconds. Access on the Ethernet terminals can be made in any order as the data sequence is independent of the physical order. It can perform Broadcast, Multicast and between slaves communications.

The EtherCAT protocol allows a precise synchronization, that is required, for example, in applications where several axis simultaneously perform coordinated movements, it can be done through an exact adjust of the *Distributed Clock*. There's also the possibility to configure devices that, as opposed to synchronous communication, have an elevated tolerance degree inside the communication system.

The configuration of EtherCAT modules is initially determined by the *Device Description Files* of the Master and Slave devices used, and can be modified by the user in the *Configuration Editor* dialog boxes. However, for conventional applications and with the desire of an as easy as possible manipulation, large-scale configurations can be automated by choosing the *Autoconfig* mode in EtherCAT Master Parameters.

Note the possibility of modifying the Master and Slave configuration parameters also in operational mode, through the Master and Slave instances, according to the availability of the device in question.

5.5.12.1. Installing and inserting EtherCAT Devices

In order to be able to insert and configure EtherCAT devices as objects in the device tree, the Slave devices must be installed.

The Master device is automatically installed by the default MasterTool IEC XE installation. The EtherCAT Master defines which Slaves can be inserted.

To install the Slave devices the *Device Repository* must be opened, use the *EtherCAT XML Device description Configuration File (*.xml)* filter and select the device description files (*EtherCAT XML Device Description / ESI EtherCAT Slave Information*), supplied with the hardware. The Slave descriptions are available as XML files (file type: *.xml).

An EtherCAT Master can be added to the *Devices Tree* through the *Add Device* command, through the context menu of the CPU NET connectors.

Under a master, one or more slaves can be added, selecting an EtherCAT Master and running the *Add Device* command (context menu of the EtherCAT Master) or running the *Scan For Devices* command.

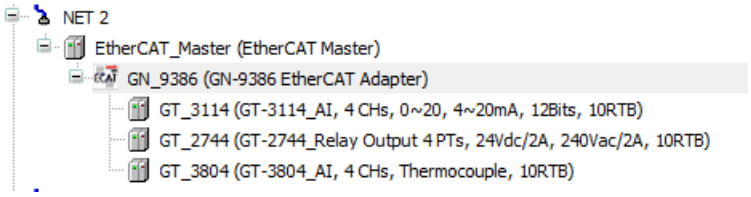


Figure 93: EtherCAT Configuration Example

ATTENTION

- Only one EtherCAT Master instance per project is allowed.
- Only available on the NET connectors of the PLC.
- It cannot be used when the NETs are set as redundant.
- It cannot be used when Project has cluster redundancy.
- Other drivers cannot be instanced in the same NET port as the EtherCAT Master.

5.5.12.1.1. EtherCAT - Scan For Devices

The *Scan For Devices* command, available in the EtherCAT Master context menu, runs a search for the Slave devices physically installed in the EtherCAT network of the PLC currently connected. This means that with this command it's possible to detect and visualize the hardware components in the window presented in the figure below, allowing the user to map them directly in the project *Device Tree* do projeto.

It's noteworthy that, when the *Scan For Devices* command is selected, a connection with the PLC will be automatically established before the search begins and terminated when the search ends. So, for the first execution of this command, the Gateway connection must be configured and a program with the EtherCAT Master configured must be loaded into the PLC. In case of future additions of Slave devices, in order to run this command, it's necessary that the EtherCAT network is stopped. To do this, put to TRUE the *bStopBus* bit, present in the variables of the EtherCAT Master Diagnostics.

When the command is executed, the *Scanned Devices* field will contain a list of all devices and modules found during the last scan. To add them to the project, just click on the button *Copy All Devices To Project*. It's also possible to perform a comparison of the devices found in the search with the ones in the project by selecting the box *Show differences to project*.

If you add an EtherCAT Master module to the Project and use the *Scan For Devices* command, you will have a list of all the available EtherCAT Slaves. Entries in bold will be shown, if there's more than one device with the same description. With a double click on the entrance a list will open, and so the desired device can be selected.

After completing the changes in the EtherCAT network configuration, it's necessary to do a new project download, for the changes to take effect.

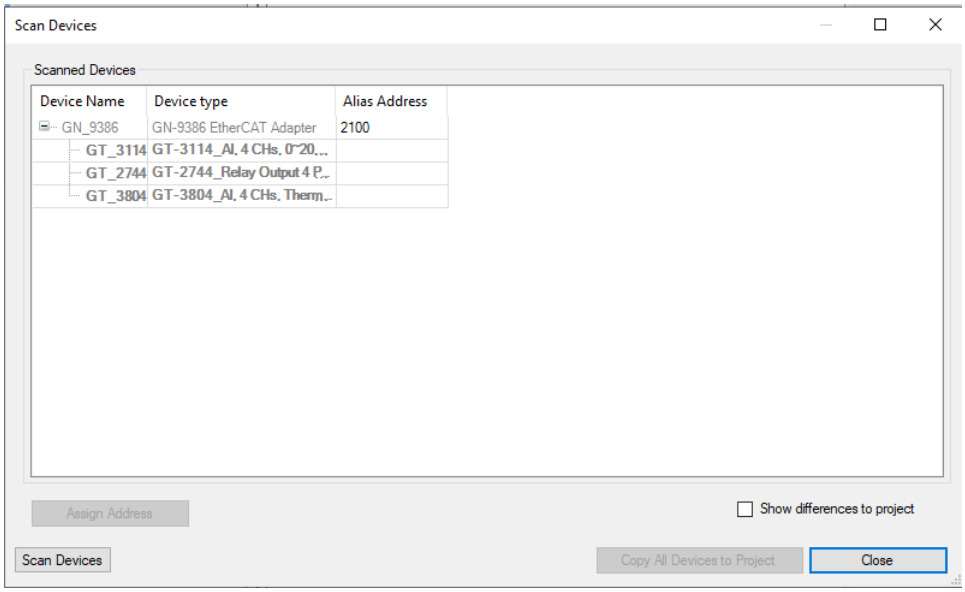


Figure 94: EtherCAT Devices Search Dialog

5.5.12.2. EtherCAT Master Settings

Below are listed the options to carry out the EtherCAT Master configuration, such as defined in *Device Description File*.

5.5.12.2.1. EtherCAT Master Parameters

Below are the general parameters found in the initial screen of the EtherCAT Master configuration, according figure below.

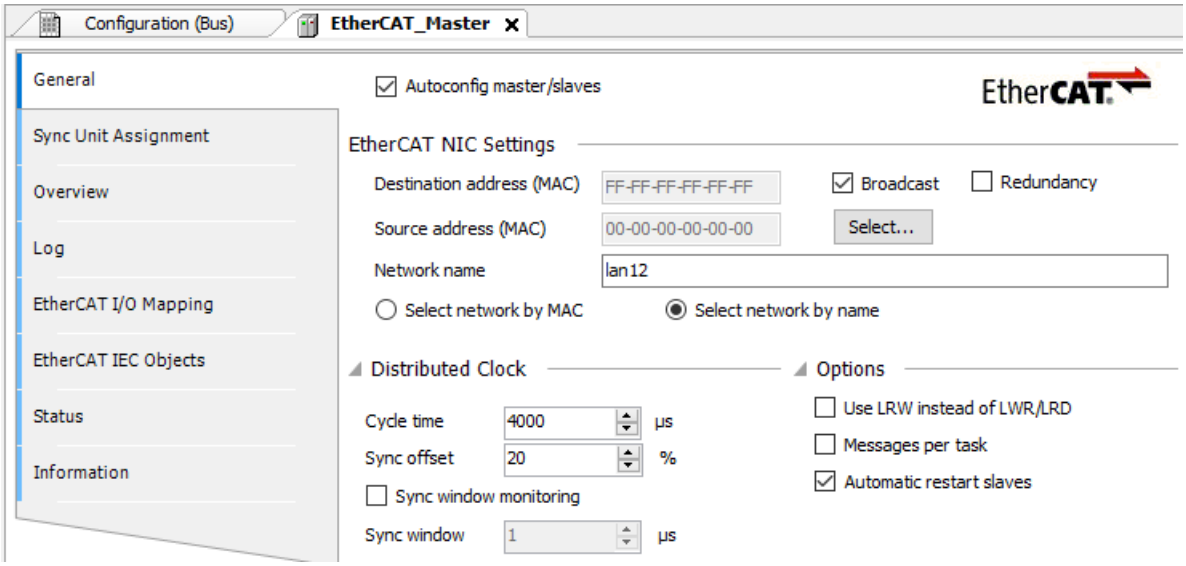


Figure 95: EtherCAT Master Configuration Dialog

Device Configuration	Description	Factory Default	Possible Values
Autoconfig master/slaves	Enable the Master and Slave automatic configuration.	Marked	Marked Unmarked
Cycle time [μs]	Sets the time period in which a new data telegram must be send to the bus.	4000	2000 to 1000000
Sync Offset [%]	Adjust the offset, from the PLC cycle, of the EtherCAT Slave synchronization interrupt.	20	-50 to 50
Sync window monitoring	If enabled, this option allows monitoring the Slave synchronization.	Unmarked	Marked Unmarked
Sync window [μs]	Time for the Sync Window Monitoring.	1	1 to 32768
Use LRW instead of LWR/LRD	Enabling of the combined read and write commands.	Unmarked	Marked Unmarked
Messages per task	If enabled, the read and write commands that are dealing with input and output messages can be done in different tasks.	Unmarked	Marked Unmarked

Device Configuration	Description	Factory Default	Possible Values
Automatic restart slaves	Restart the devices when the communication is aborted.	Marked	Marked Unmarked

Table 135: EtherCAT Master Configuration

Notes:

Autoconfig master/slaves: If this option is enabled, most of Master and Slave configuration will be made automatically, based on the description files and implicit calculations. In this case, the FMMU / Sync dialog will not be available. If it's unchecked the *Image In Address* and *Image Out Address* options will be available to the user.

ATTENTION

The *Autoconfig* mode is enabled by default and usually enough and highly recommended for standard applications. If it's disabled, all configuration definitions will have to be made manually, and thus, some specialized knowledge is required. To configure a Slave-to-Slave communication, the *Autoconfig* option must be disabled.

Cycle time: Time period after which, a new data telegram must be sent to the bus. If *Distributed Clock* functionality is enabled, the value of this parameter will be transferred to the Slaves clocks. This way, a precise data exchange synchronization can be achieved, which is especially important in cases where the distributed process demands simultaneous actions. So, a very precise time base, with a jitter significantly smaller than a microsecond, for all the network can be achieved.

Sync Offset: This value allows the adjustment of the offset of the EtherCAT Slave synchronization interrupt to the PLC cycle. Normally, the PLC task cycle begins 20% later than the Slaves synchronization interruption. This means that the PLC task can be delayed by 80% of the cycle time and no message will be lost.

Sync Window: If the synchronization of all Slaves are inside this time window, the EtherCAT Master *bDistributedClockInSync* diagnostic will be set to TRUE, otherwise it will be set to FALSE. When Distributed Clock is used, it's highly recommended to use a dedicated task with high priority as the *Bus cycle task* of the EtherCAT Master. To do this, it's necessary to use Project Profiles that allows the creation of new tasks, then create a cyclic task with priority 0 (real time task) and link it to the master *Bus cycle task* on the EtherCAT Master - I/O Mapping tab of the EtherCAT Master. The user can also change the value of the *wDCInSyncWindow* variable, configuring the maximum jitter allowed on the synchronization between master and slaves.

Use LRW instead of LWR/LRD: Activating this option enables the Slave-to-Slave communication because, instead of using separated reading (LRD) and write (LWR) commands, combined reading/writing (LRW) commands will be used.

Automatic Restart Slaves: By enabling this option, the Master will restart the Slaves as soon as the communication is aborted.

5.5.12.2.2. EtherCAT Master - Sync Unit Assignment

This tab of the EtherCAT Master configuration editor shows all slaves that are entered below a specific master with an assignment to the sync units.

With EtherCAT sync units, multiple slaves are configured into groups and subdivided into smaller units. For each group, the job counter can be monitored for better and more accurate error detection. As soon as a slave is missing from a group of synchronization units, the other slaves in the group are also shown as missing. Detection occurs immediately on the next bus cycle because the job counter is checked continuously. With device diagnostics, the missing group can be remedied as quickly as possible.

Unaffected groups remain operable without any interference.

5.5.12.2.3. EtherCAT Master - Overview

This tab of the EtherCAT Master configuration editor provides an overview of the states of all slaves, which are entered below this master and have an address. Modules are not displayed.

5.5.12.2.4. EtherCAT Master - I/O Mapping

This EtherCAT Master configuration editor tab offers the possibility to change the task that will be used for bus updates.

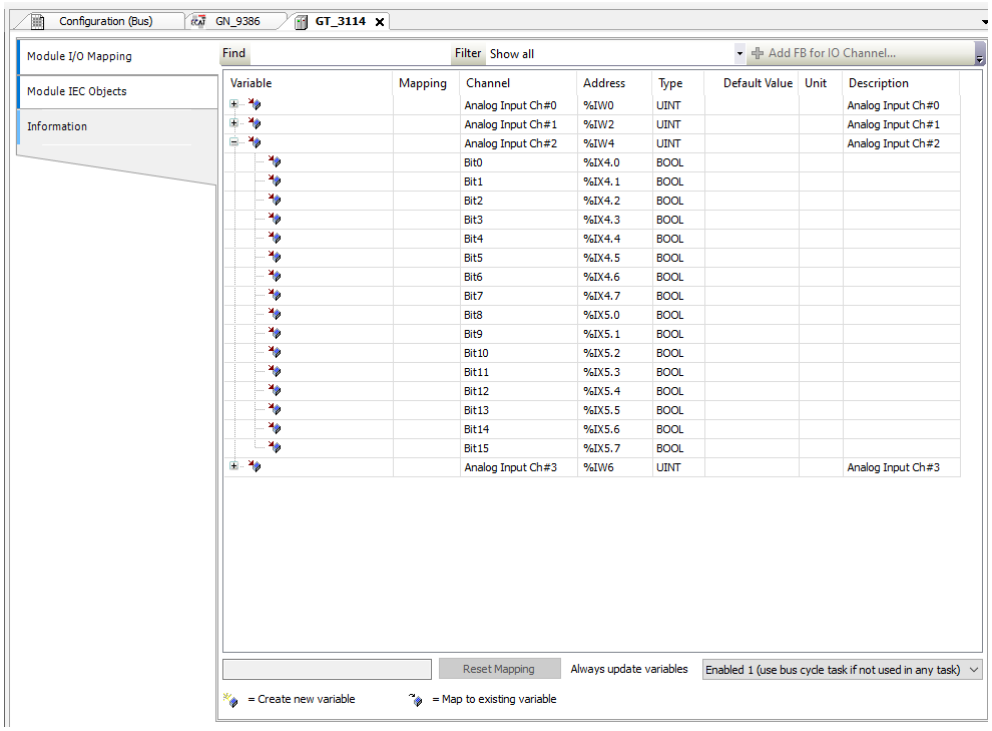


Figure 96: Slave I/O Mapping Dialog

5.5.12.2.5. EtherCAT Master - Status / Information Tabs

The Status tab of the EtherCAT Master configuration editor provides status information (e.g. '*Running*', '*Stopped*') and diagnostic messages specific of the device and the internal bus system.

The Information tab, present on the EtherCAT Master configuration editor, shows, if available, the following general information about the module: *Name, Vendor, Type, Version Number, Category, Order Number, Description, Image*.

5.5.12.3. EtherCAT Slave Configuration

Below are listed the main EtherCAT Slave configuration options, as defined in the *Device Description File*.

5.5.12.3.1. EtherCAT Slave - General

Below are presented the general parameters found in EtherCAT Slave configuration initial screen. This field is only available if the *Autoconfig* mode (Master) isn't enabled.

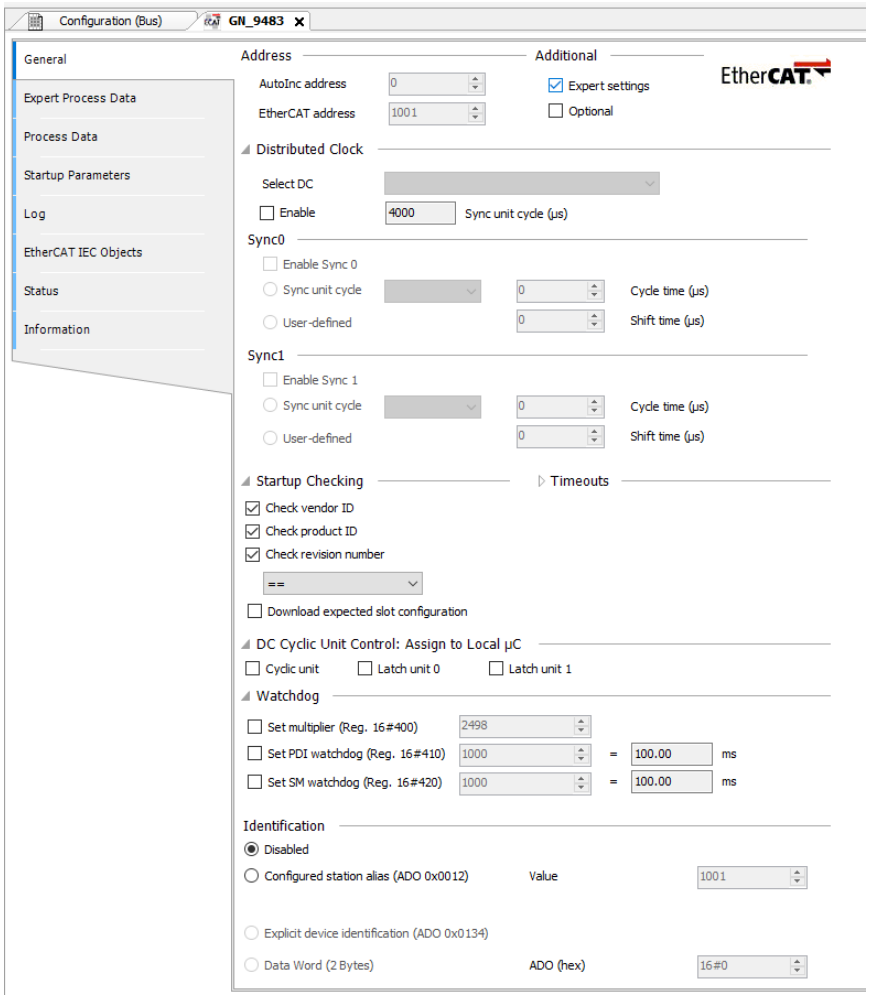


Figure 97: EtherCAT Slave Configuration Dialog

Device Configuration	Description	Default Value	Options
AutoInc Address	Auto incremental Address (16-bit) defined by the Slave position in the network.	-	-65535 to 0
EtherCAT Address	Slave final address, assign by the Master during startup. This address is independent from the position in the network.	-	1 to 65535
Expert settings	Enable the Slave advanced Settings options.	Unmarked	Marked Unmarked
Optional	Declare the Slave as Optional.	Unmarked	Marked Unmarked
Select DC	Show all Distributed Clock configurations provided by the device description file.	-	-
Enable Distributed Clock	Enable the Distributed Clock configuration options.	Unmarked	Marked Unmarked

Device Configuration	Description	Default Value	Options
Sync Unit Cycle [μs]	Show the Cycle Time set in Master.	100000	2000 to 1000000
Enable (Sync 0)	Enable the Sync 0 synchronization unit configurations.	Unmarked	Marked Unmarked
Sync Unit Cycle (Sync 0)	By selecting this option, the Cycle Time will be determined by the product of the factor and the Sync Unit Cycle.	Unmarked	Marked Unmarked
User Defined (Sync 0)	If this option is selected, the desired time, in microseconds, can be directly set into the Cycle Time (μs) field.	Unmarked	Marked Unmarked
Cycle Time [μs] (Sync 0)	Show the cycle time currently set.	100000	1 to 2147483647
Shift Time [μs] (Sync 0)	Time between the sync events and the "Output Valid" or "Input Latch" time.	0	-2147483648 to 2147483647
Enable (Sync 1)	Enable the Sync 1 synchronization unit configurations.	Unmarked	Marked Unmarked
Sync Unit Cycle (Sync 1)	By selecting this option, the Cycle Time will be determined by the product of the factor and the Sync Unit Cycle.	Unmarked	Marked Unmarked
User Defined (Sync 1)	If this option is selected, the desired time, in microseconds, can be directly set into the Cycle Time (μs) field.	Unmarked	Marked Unmarked
Cycle Time [μs] (Sync 1)	Show the cycle time currently set.	100000	1 to 2147483647
Shift Time [μs] (Sync 1)	Time between the sync events and the "Output Valid" or "Input Latch" time.	0	-2147483648 to 2147483647
Check Vendor ID	If unmarked, it will disable the Vendor ID Check.	Marked	Marked Unmarked
Check Product ID	If unmarked, it will disable the Product ID Check.	Marked	Marked Unmarked
SDO Access	Set a time reference for the timeout check of a SDO Access.	-	0 to 100000
I -> P	Set a time reference for the timeout check of the switch from Init to Pre-Operation mode.	-	0 to 100000

Device Configuration	Description	Default Value	Options
P -> S/S -> O	Set a time reference for the timeout check of the switch from Pre-Operation to Safe-Operation and from Safe-Operation to Operational modes.	-	0 to 100000
Cyclic Unit	Set the Unit Cycle to the local microprocessor.	Unmarked	Marked Unmarked
Latch Unit 0	Set the Latch Unit 0 to the local microprocessor.	Unmarked	Marked Unmarked
Latch Unit 1	Set the Latch Unit 1 to the local microprocessor.	Unmarked	Marked Unmarked

Table 136: EtherCAT Slave Configurations

Notes:

AutoInc Address: This address is used only during startup, when the Master is assigning the EtherCAT addresses to the Slaves. When for this matter, the first telegram runs through the Slaves, each fast-read Slave increases its *AutoInc* Address by 1. The Slave with address 0 finally will receive the data.

Optional: If a Slave is declared as *Optional*, no error message will be created in case the device doesn't exist in the bus system. Thus a *Station alias* address must be defined and written to the EEPROM. This option is only available if the option *Autoconfig Master/Slaves* in the settings of the EtherCAT Master is activated and if this function is supported by the EtherCAT Slave.

Enable Distributed Clock: If the *Distributed Clock* functionality is enabled, the data exchange cycle time, displayed in the *Sync Unit Cycle (µs)* field will be determined by the Master *Cycle Time*. Thus the master clock can synchronize the data exchange within the network. The settings for handling the synchronization unit(s) depend on the Slave.

Enable Sync 0: If this option is activated, the *Sync0* synchronization unit is used. A synchronization unit describes a set of process data which is exchanged synchronously.

Sync Unit Cycle (Sync 0): If this option is activated, the Master *Cycle Time*, multiplied by the chosen factor will be used as synchronization cycle time for the slave. The *Cycle Time (µs)* field shows the currently set cycle time.

Shift Time: The Shift Time describes the time between the sync events (*Sync0, Sync1*) and the *Output Valid* or *Input Latch* times. Writable value, if the slave supports shifting of *Output Valid* or *Input Latch*.

Enable Sync 1: If this option is selected, the synchronization unit *Sync1* is used. A synchronization unit is a set of process data which is exchange synchronously.

Sync Unit Cycle (Sync1): If this option is activated, the Master *Cycle Time*, multiplied by the chosen factor will be used as synchronization cycle time for the slave. The *Cycle Time (µs)* field shows the currently set cycle time.

Check Vendor ID and Product ID: By default, at startup of the system the *Vendor ID* and/or the *Product ID* will be checked against the current configured settings. If a mismatch is detected, the bus will be stopped and no further actions will be executed. This serves to avoid the download of an erroneous configuration. This option is intended to switch off the check, if necessary.

SDO Access: By default there's no timeout set for the SDO list submit action at system startup. However, if it's necessary to check if this action exceeds a certain time, it must be defined (in microseconds) in this field.

I -> P: By default there's no timeout set for the state transition from *Init* to *Pre-Operational*. However, if it's necessary to check if this action exceeds a certain time, it must be defined (in microseconds) in this field.

P -> S / S -> O: By default there's no timeout set for the state transition from *Pre-Operational* to *Safe-Operational* and from *Safe-Operational* to *Operational*. However, if it's necessary to check if this action exceeds a certain time, it must be defined (in microseconds) in this field.

DC cycle unit control: Choose the desired option(s) concerning the *Distributed Clock* functions in order to define, which should be assigned to the local microprocessor. The control is done in register 0x980 in the EtherCAT slave. The possible settings: *Cyclic Unit, Latch Unit 0, Latch Unit 1*.

Enable: If the setting Optional is not activated, this setting can be activated if explicitly supported by the device description of the slave. It allows direct assignment of an alias address in order to get the slaves address independent of its position within the bus. If the option *Optional* is activated, this checkbox is disabled.

5.5.12.3.2. *EtherCAT Slave - Process Data*

The *Process Data* tab of the EtherCAT Slave configurator editor shows the slave input and output process data, each defined by name, type and index by the device description file, as seen in figure below.

The selected input (to be read) and output (to be written) of the device are available in the EtherCAT Slave - I/O Mapping dialog as PLC inputs and outputs to which project variables might be mapped.

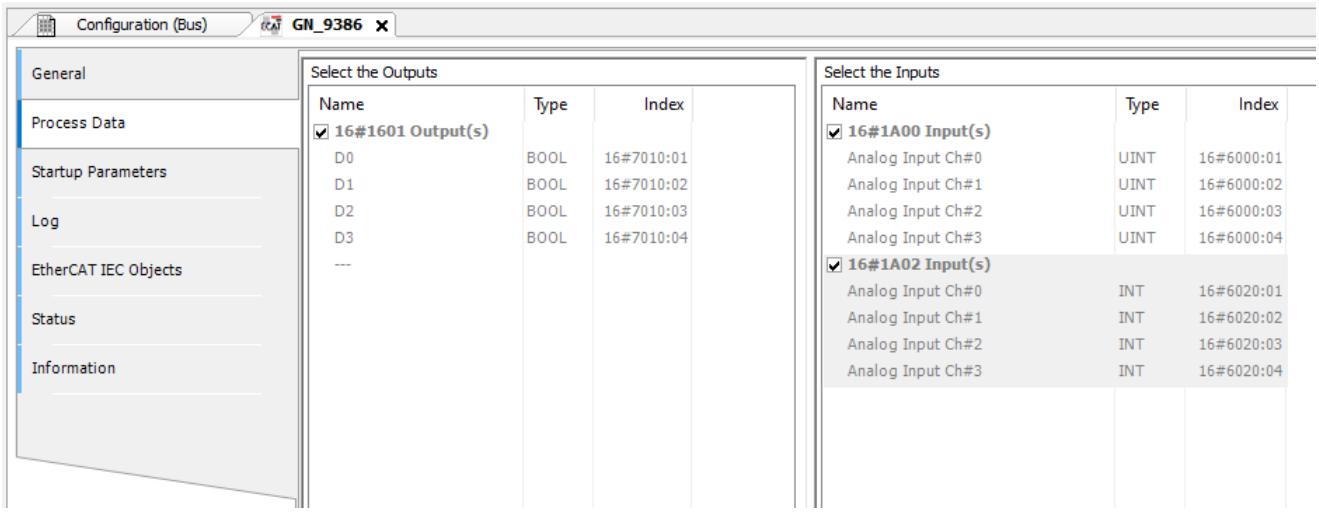


Figure 98: Process Data Dialog

The *Expert Process Data* dialog will only be available in the EtherCAT Slave configuration editor if the *Enable Expert Settings* option is activated. It provides another, more detailed, vision of the process data, adding to what is presented in the *Process Data* tab. Furthermore, the download of the *PDO Assignment* and the *PDO Configuration* can be activated in this dialog.

ATTENTION

If the Slave doesn't accept the PDO Configuration, it will stay in Pre-Operational state and none real time data exchange will be possible.

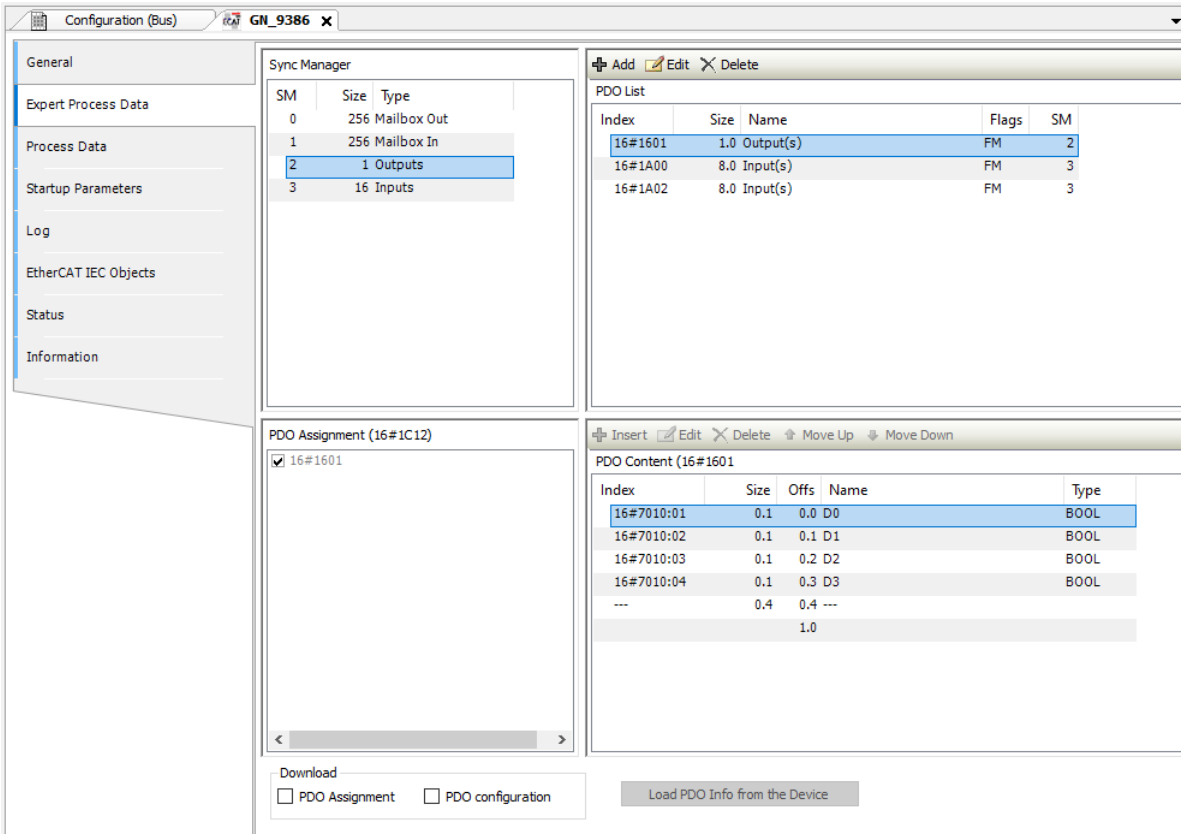


Figure 99: Expert Process Data Dialog

This dialog is divided in four sections and two options:

- *Sync Manager*: List of *Sync Manager* with data size and type of PDOs.
- *PDO Assignment*: List of PDOs assigned to the selected *Sync Manager*. The checkbox activates the PDO and I/O channels are created. It is similar to the simple PDO configuration windows. Here only PDOs can be enabled or disabled.
- *PDO List*: List of all PDOs defined in the device description file. Single PDOs can be deleted, edited or added by executing of the respective command from the context menu.
- *PDO Content*: Displays the content of the PDO selected in the section above. Entries can be deleted, edited or added by executing of the respective command from the context menu.
- *PDO Assignment*: If activated a CoE write command will be added to index 0x1CXX to write the PDO configuration 0x16XX or 0x1A00.
- *PDO Configuration*: If activated several CoE write commands will be added to write the PDO mapping to the slave.

ATTENTION

If a Slave doesn't support the PDO configuration, a download may result in a Slave error. This function should only be used by experts.

5.5.12.3.3. EtherCAT Slave - Edit PDO List

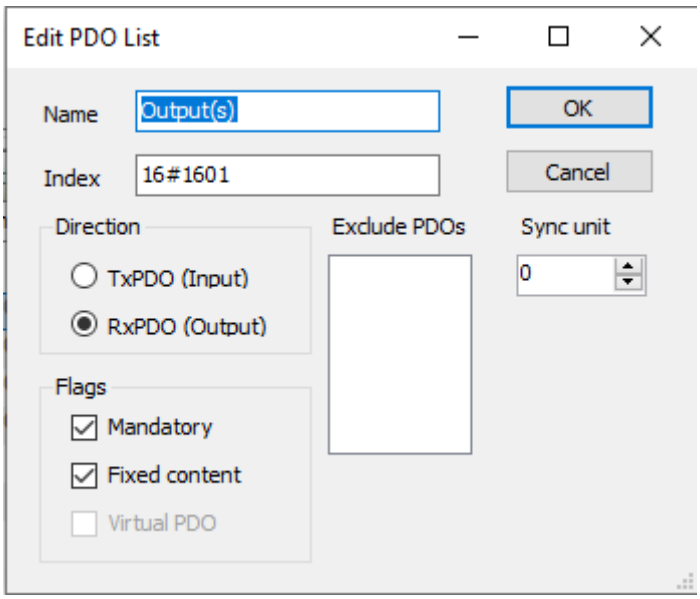


Figure 100: Edit PDO List Dialog

This dialog is opened through the context menu from the PDO List area, presented in Figure 99. Below are some explanations on the configuration options presented in this dialog.

- *Name:* Name of the PDO input.
- *Index:* Index of the PDO in being edited.
- *TxPDO (Input):* If activated, the PDO will be transferred from the Master to the Slave.
- *RxPDO (Output):* If activated, the PDO will be transferred from the Slave to the Master.
- *Mandatory:* The PDO is necessary and can't be unchecked in the *PDO Assignment* area.
- *Fixed Content:* The PDO content is fixed and can't be changed. It's not possible to add entries in the *PDO Content* panel.
- *Virtual PDO:* Reserved for future use.
- *Exclude PDOs:* It's possible to define a list of PDO that can, or can't, be selected along with the PDO being edited in the *PDO Assignment* area, or in the *Process Data* tab. If a PDO is marked in this list, it can't be selected, turning into gray in the *PDO Assignment* area when the PDO in edition is selected.
- *SyncUnit:* ID of the PDO *Sync Manager* shall assigned to.

5.5.12.3.4. EtherCAT Slave - Startup Parameters

In the *Startup Parameters* tab, parameters for the device can be defined, which will be transferred by SDOs (*Service Data Objects*) or IDN at the system's startup. The options available in this tab, as well as the access possibilities, vary according to the EtherCAT Slave used and they are present in the *Device Description File*.

5.5.12.3.5. EtherCAT Slave - I/O Mapping

This tab of the EtherCAT Slave configuration editor offers the possibility to assign the project variables to the EtherCAT inputs or outputs. This way, the EtherCAT Slave variables can be controlled by the *User Application*.

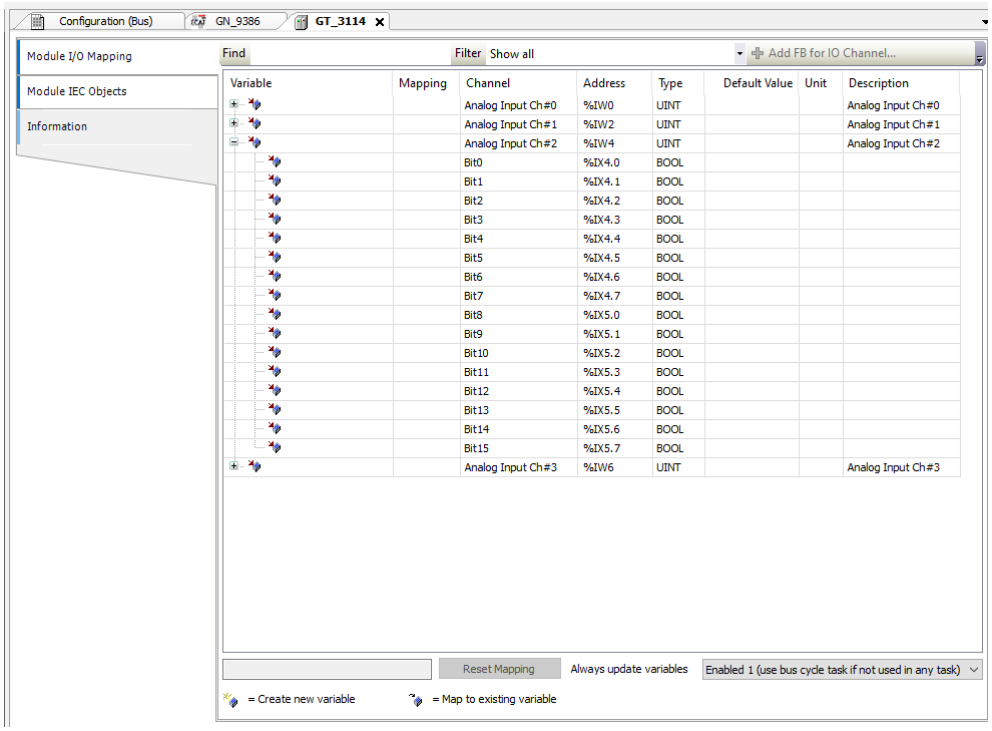


Figure 101: Slave I/O Mapping Dialog

5.5.12.3.6. EtherCAT Slave - Status and Information

The *Status* tab of the EtherCAT Slave provides status information (e.g. '*Running*', '*Stopped*') and device-specific diagnostic messages, also on the used card and the internal bus system.

The *Information* tab, presented in the EtherCAT Slave configuration editor, shows, if available, the following general information about the module: *Name, Vendor, Type, Version, Categories, Order Number, Description, Image*.

5.5.13. EtherNet/IP

The EtherNet/IP is a master-slave architecture protocol, consisting of an EtherNet/IP Scanner (master) and one or more EtherNet/IP Adapters (slave).

The Ethernet/IP protocol is based on CIP (*Common Industrial Protocol*), which have two primary purposes: The transport of control-oriented data associated with I/O devices and other system-related information to be controlled, such as configuration parameters and diagnostics. The first one is done through implicit messages, while the second one is done through explicit messages.

Their runtime system can act as either Scanner or Adapter. Each CPU's NET interface support only one EtherNet/IP instance and it can't be instanced on an Ethernet expansion module.

An EtherNet/IP Adapter instance supports an unlimited number of modules or Input/Output bytes. In these modules, can be added variables of types: BYTE, BOOL, WORD, DWORD, LWORD, USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL and LREAL.

ATTENTION

EtherNet/IP can't be used together with Ethernet Redundant Mode or with Half-Cluster's redundancy.

ATTENTION

To avoid communication issues, EtherNet/IP Scanner can only have Adapters configured within the same subnetwork.

5.5.13.1. EtherNet/IP

To add an EtherNet/IP Scanner or Adapter it's needed to add an *Ethernet Adapter* under the desired NET. This can be done through the command *Add Device*. Under this *Ethernet Adapter* it's possible to add a *Scanner* or an *Adapter*.

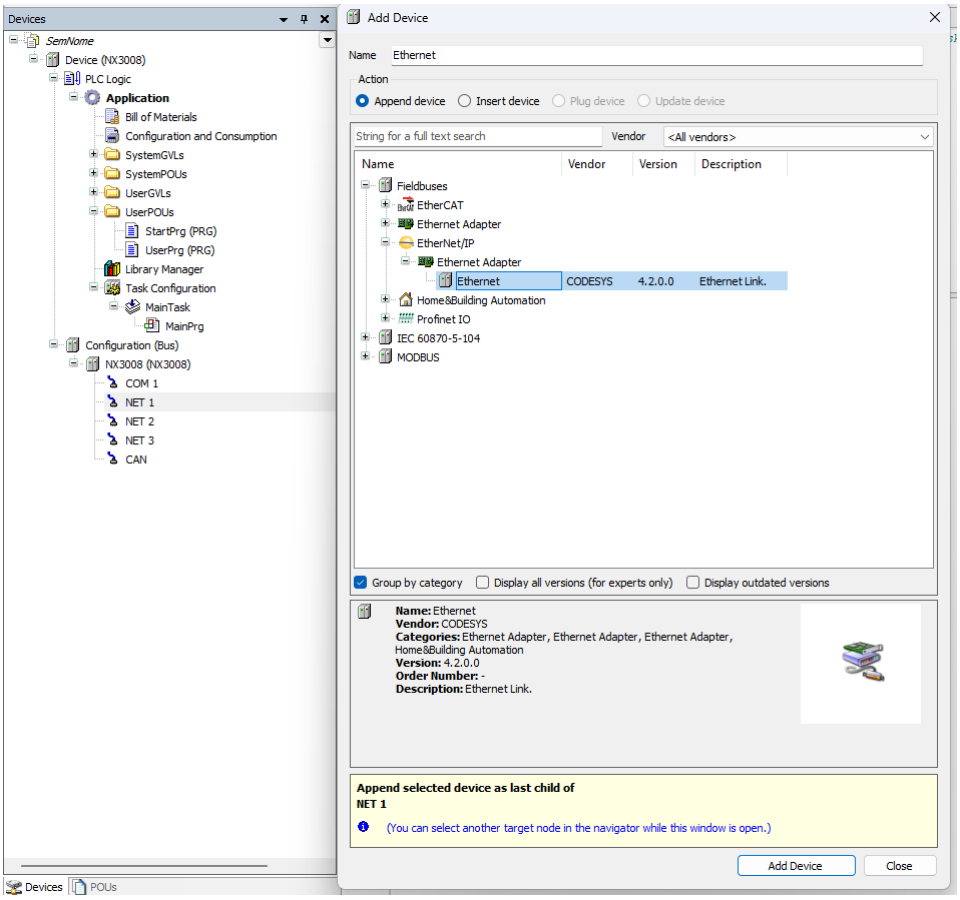


Figure 102: Adding an Ethernet Adapter

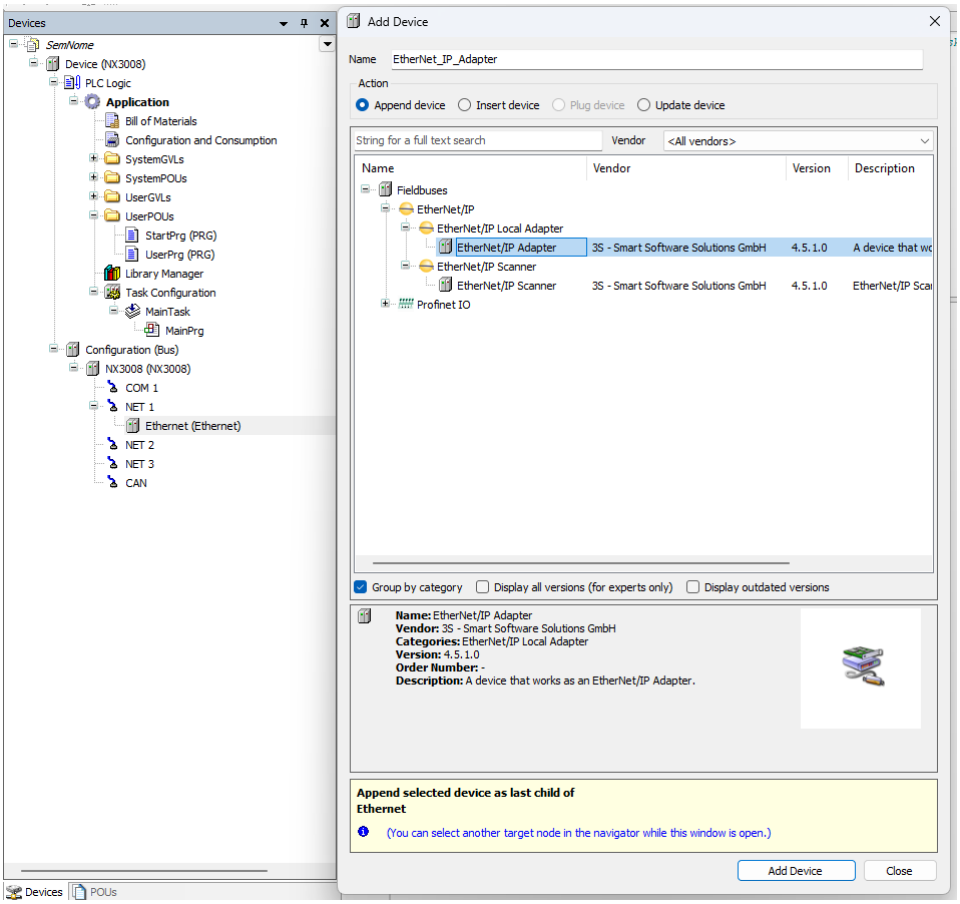


Figure 103: Adding an EtherNet/IP Adapter or Scanner

5.5.13.2. EtherNet/IP Scanner Configuration

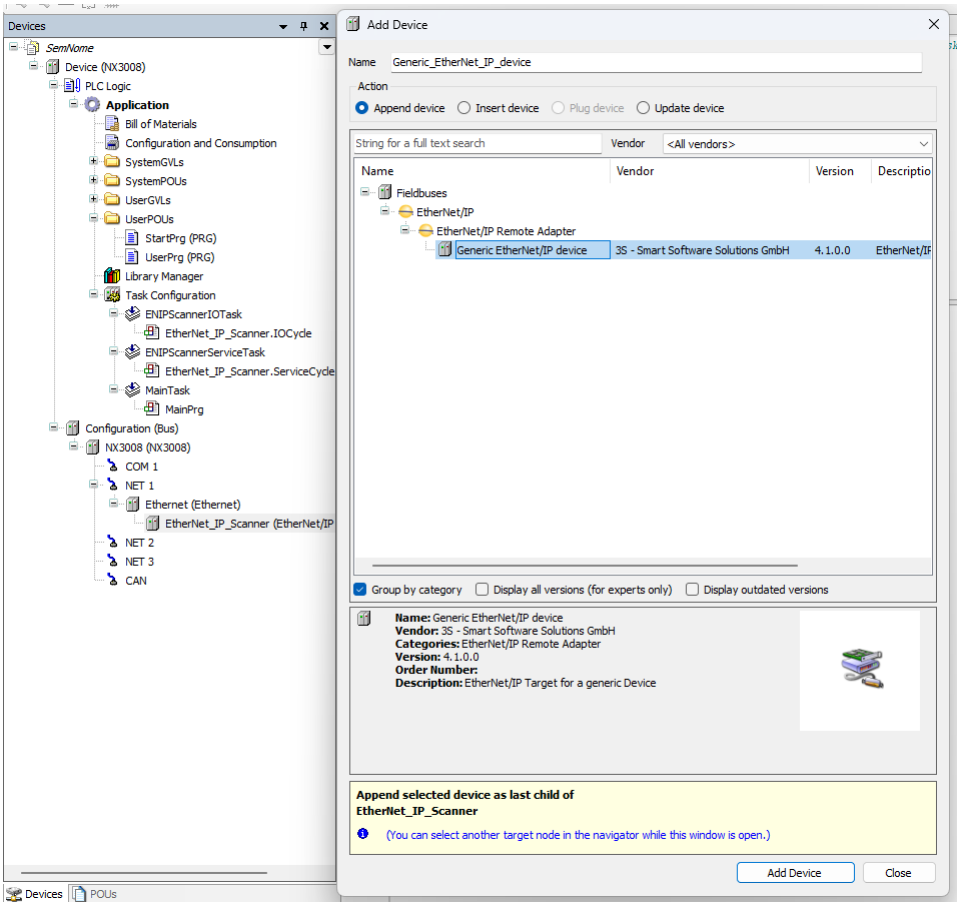


Figure 104: Adding an EtherNet/IP Adapter Under the Scanner

5.5.13.2.1. General

After open the Adapter declared under the Scanner it's possible to configure it as needed. The first Tab is *General*, on it is possible to configure the *IP address* and the *Electronic Keying* parameters. These parameters must be checked or unchecked if the adapter being used is installed on MasterTool. Otherwise, if the Adapter used is of type Generic. The Vendor ID, Device Type, Product Code, Large Revision, and Small Revision fields must be filled in with the correct vendor's information and the boxes checked as much as necessary. Altus, for its part, has its own ID, which is "1454".

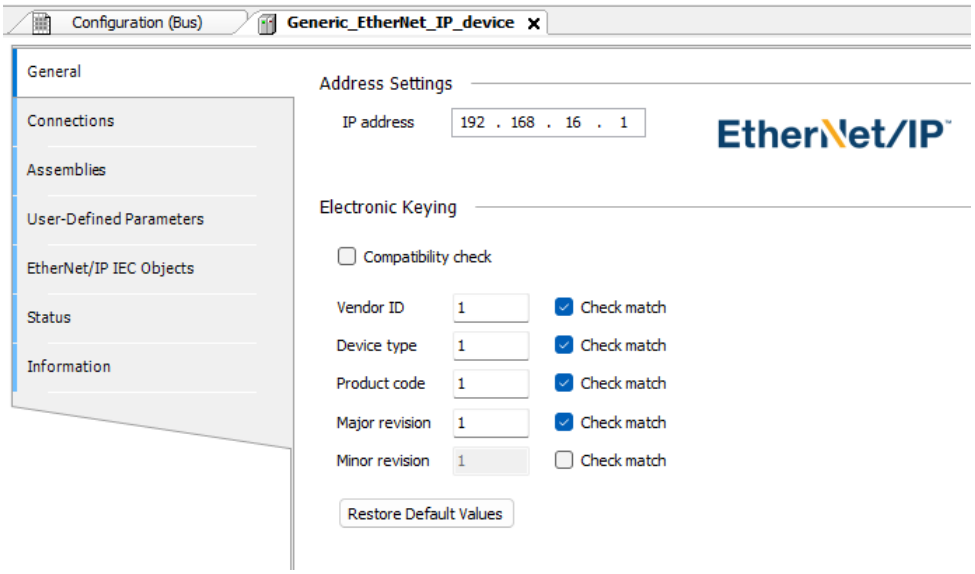


Figure 105: EtherNet/IP General Tab

5.5.13.2.2. *Connections*

The upper area of the *Connections* tab displays a list of all configured connections. When there is an *Exclusive Owner* connection in the EDS file, it is inserted automatically when the Adapter is added. The configuration data for these connections can be changed in the lower part of the view.

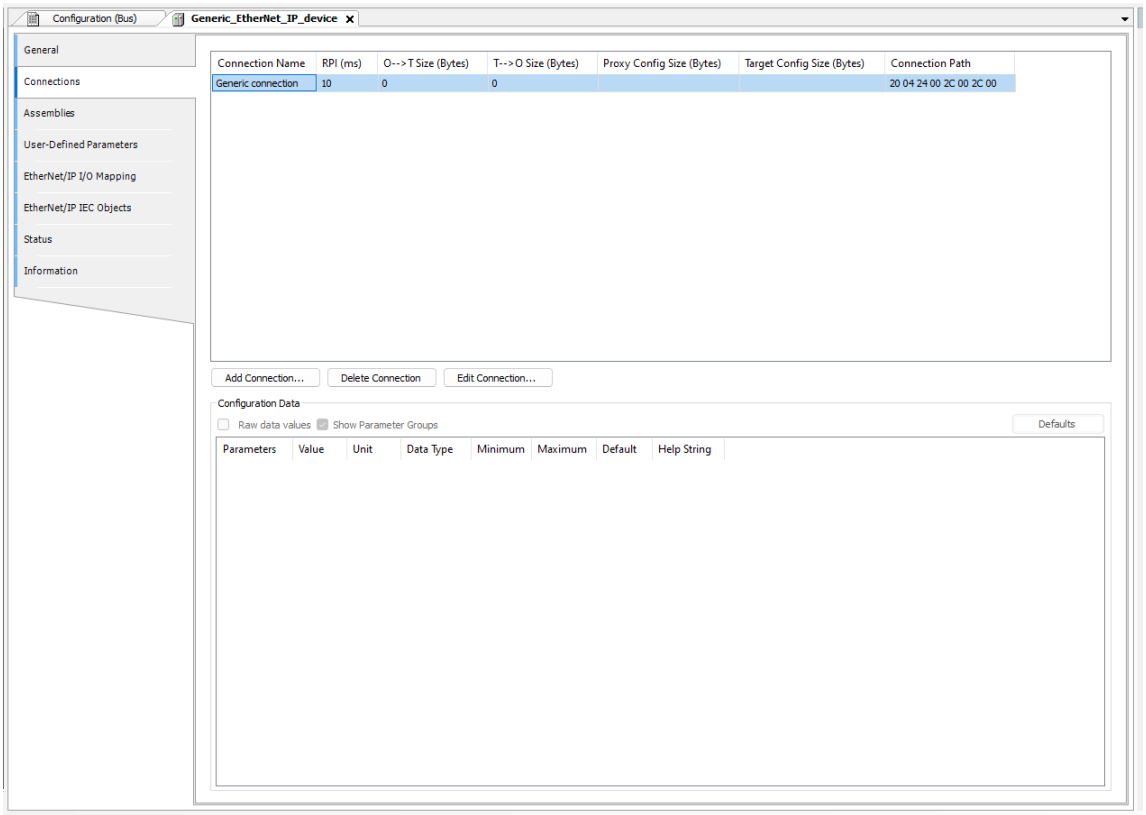


Figure 106: EtherNet/IP Connection Tab

Notes:

For two or more EtherNet/IP Scanners to connect to the same Remote Adapter:

1. Only one of the Scanners can establish an *Exclusive Owner* connection.
2. The same value of *RPI(ms)* must be configured for the Scanners.

The configuration data is defined in the EDS file. The data is transmitted to the remote adapter when the connection is opened.

Configuration	Description	Default Value	Options
RPI (ms)	Request Packet Interval: exchange interval of the input and output data.	10 ms	Multiple the Interval of the Bus Cycle Task to which it is associated
O -> T Size (Bytes)	Size of the producer data from the Scanner to the Adapter (O -> T)	0	0 - 65527
T -> O Size (Bytes)	Size of the consumer data from the Adapter to the Scanner (T -> O)	0	0 - 65531
Proxy Config Size (Bytes)	Proxy configuration data size	-	-
Device Config Size (Bytes)	Device configuration data size.	-	-
Connection Path	Address of the configuration objects - input objects - output objects.	Automatically generated path	Automatically generated path, User-defined path and Path defined by symbolic name

Table 137: EtherNet/IP Connection parameters

To *add* new connections there is the button *Add Connection...* which will open the *New connection* window. In this window, you can configure a new connection type from those predefined in the Adapter's EDS or a connection from zero when using a Generic device.

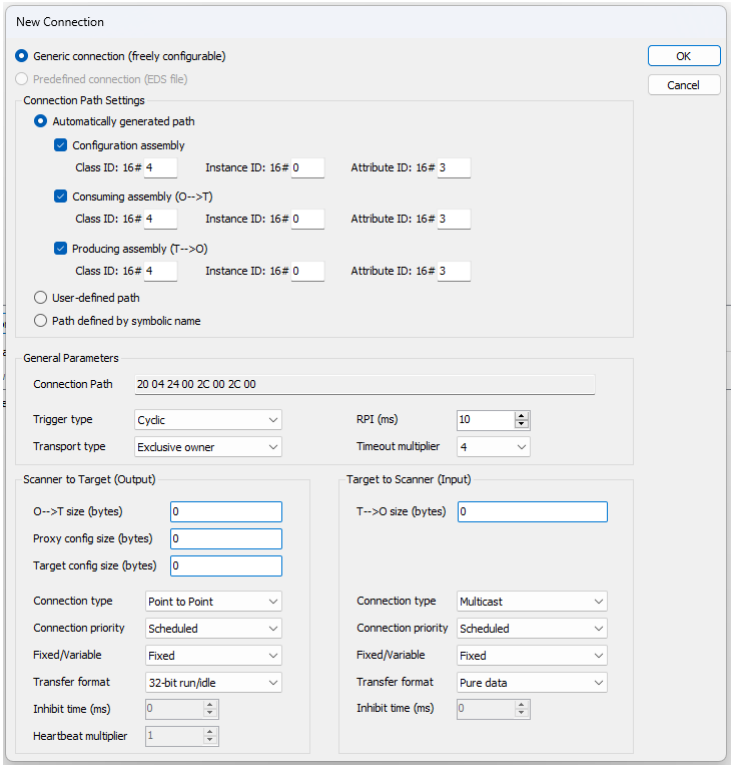


Figure 107: EtherNet/IP New Connection's Window

5.5.13.2.3. *Assemblies*

The upper area of the *Assemblies* tab displays a list of all configured connections. When a connection is selected, the associated inputs and outputs are displayed in the lower area of the tab.

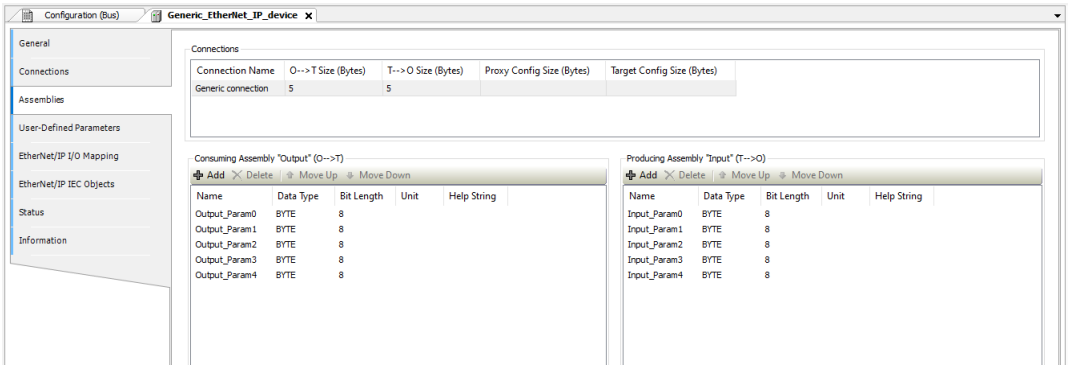


Figure 108: EtherNet/IP Assemblies

Output Assembly and *Input Assembly*:

Configuration	Description
Add	Opens the dialog box "Add Input/Output"
Delete	Deletes all selected Inputs/Outputs.
Move Up	Moves the selected Input/Output within the list.
Move Down	The order in the list determines the order in the I/O mapping.

Table 138: EtherNet/IP Assemblies tab

Dialog box *Add Input/Output*:

Configuration	Description
Name	Name of the input/output to be inserted.
Help String	
Data type	Type of the input/output to be inserted. This type also define its Bit Length.
Bit Length	This value must not be edited.

Table 139: EtherNet/IP "Add Input/Output" window

5.5.13.2.4. EtherNet/IP I/O Mapping

I/O Mapping tab shows, in the *Variable* column, the name of the automatically generated instance of the *Adapter* under *IEC Objects*. In this way, the instance can be accessed by the application. Here the project variables are mapped to adapter's inputs and outputs.

5.5.13.3. EtherNet/IP Adapter Configuration

The EtherNet/IP Adapter requires Ethernet/IP Modules. The Modules will provide I/O mappings that can be manipulated by user application through %I or %Q addresses according to its configuration.

New Adapters can be installed on MasterTool with the EDS Files. The configuration options may differ depending on the device description file of the added Adapter.

5.5.13.3.1. General

The first tab of the EtherNet/IP Adapter is the *General* tab. Here you can set the parameters of the *Electronic Keying* used in the scanner to check compatibility. In this tab, you can also install the EDS of the device directly in the MasterTool device repository or export it.

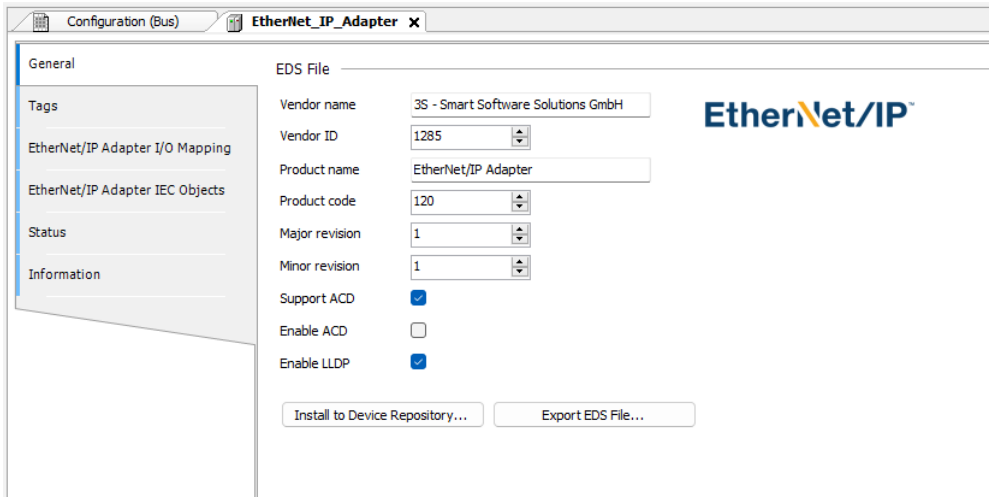


Figure 109: EtherNet/IP General Tab

5.5.13.3.2. EtherNet/IP Adapter: I/O Mapping

On the *EtherNet/IP I/O Mapping* tab, you can configure which bus cycle task the Adapter will execute.

5.5.13.4. EtherNet/IP Module Configuration

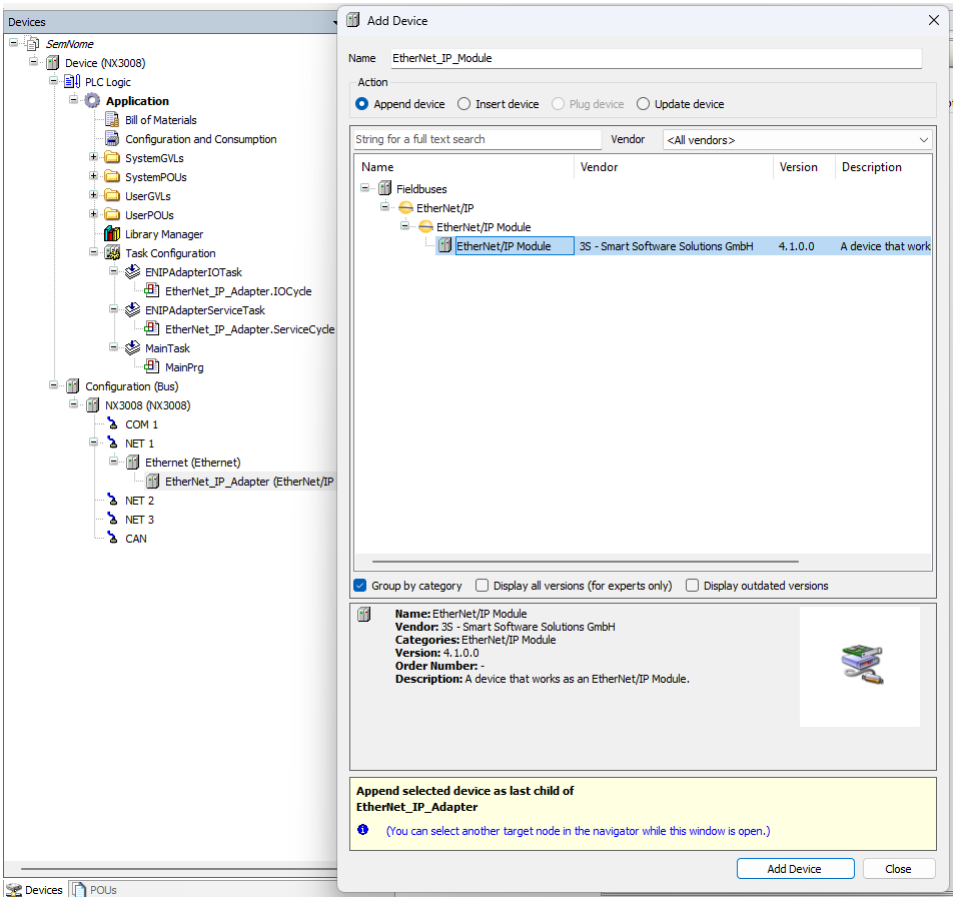


Figure 110: Adding an EtherNet/IP Module under the Adapter

5.5.13.4.1. Assemblies

The parameters of the module's *General* tab follow the same rules as described in the 138 and 139 tables.

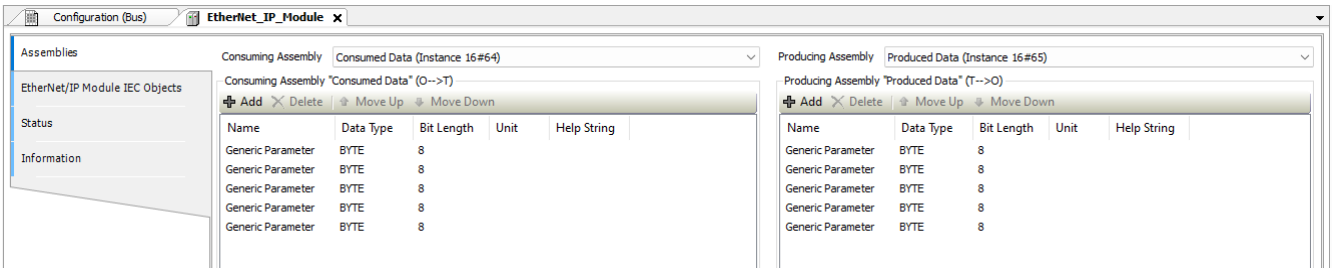


Figure 111: EtherNet/IP Module Assemblies tab

5.5.13.4.2. EtherNet/IP Module: I/O Mapping

The I/O Mapping tab shows, in the *Variable* column, the name of the automatically generated Adapter instances. In this way, the instance can be accessed by the user application.

5.5.14. IEC 60870-5-104 Server

As select this option at MasterTool, the CPU starts to be an IEC 60870-5-104 communication server, allowing connection with up to three client devices. To each client the driver owns one exclusive event queue with the following features:

- Size: 1000 events
- Retentivity: non retentive
- Overflow policy: keep the newest

To configure this protocol, it is needed to do the following steps:

- Add a protocol IEC 60870-5-104 Server instance to one of the available Ethernet channel. To realize this procedure consult the section Inserting a Protocol Instance
- Configure the Ethernet interface. To realize this procedure consult the section Ethernet Interfaces Configuration
- Configure the general parameters of protocol IEC 60870-5-104 Server with connection mode Port or IP, and the TCP port number when the selected connection mode is IP
- Add and configure devices, defining the proper parameters
- Add and configure the IEC 60870-5-104 mappings, specifying the variable name, type of object, object address, size, range, dead band and type of dead band
- Configure the link layer parameters, specifying the addresses, communication time-outs and communication parameters
- Configure the application layer parameters, synchronism configuration, commands, as well as transmission mode of *Integrated Totals* objects

The descriptions of each configuration are related below, in this section.

5.5.14.1. Type of data

The table below shows the supported variable type by the Nexto Series CPU for each protocol IEC 60870-5-104 data type.

Object Type	IEC Variables Type
Single Point Information (M_SP_NA)	BOOL
	BIT
Double Point Information (M_DP_NA)	DBP
Step Position Information (M_ST_NA)	USINT
Measured Value, normalized value (M_ME_NA)	INT
Measured Value, scaled value (M_ME_NB)	INT
Measured Value, short floating point value (M_ME_NC)	INT
	UINT
	DINT
	UDINT
	REAL
Integrated Totals (M_IT_NA)	INT
	DINT
Bitstring Information (M_BO_NA)	DWORD
Single Command (C_SC_NA)	BOOL
	BIT
Double Command (C_DC_NA)	DBP
Regulating Step Command (C_RC_NA)	DBP
Setting Point Command, normalized Value (C_SE_NA)	INT
Setting Point Command, scaled Value (C_SE_NB)	INT
Setting Point Command, short floating point Value (C_SE_NC)	REAL
Bitstring Command (C_BO_NA)	DWORD

Table 140: Variables Declaration to IEC 60870-5-104

Notes:

Regulating Step Command: The *Lower* and *Higher* object states of the C_RC_NA are associated respectively to *OFF* and *ON* internal *DBP* type states.

Step Position Information: According to item 7.3.1.5 of Standard IEC 60870-5-101, this 8 bit variable is compose by two fields: value (defined by the 7 bits less significant) and transient (defined as the most significant bit, which indicates when the measured device is transitioning).

Below, there is a code example for fields manipulation in an USINT type variable. Attention, because this code doesn't consist if the value is inside the range, therefore this consistency remains at user's charge.

```
PROGRAM UserPrg
VAR
usiVTI: USINT;  // Value with transient state indication, mapped for the Client
siValue: SINT;  // Value to be converted to VTI. Must be between -64 and +63
bTransient: BOOL; // Transient to be converted to VTI
END_VAR

usiVTI := SINT_TO_USINT(siValue) AND 16#3F;
IF siValue < 0 THEN
usiVTI := usiVTI OR 16#40;
END_IF
IF bTransient THEN
usiVTI := usiVTI OR 16#80;
END_IF
```

PROFIBUS: Except by the digital objects, the protocol IEC 60870-5-104's analog and counters objects data types are different from PROFIBUS analogs and counters modules data types, not being possible to map such PROFIBUS variable types directly to IEC 60870-5-104 clients.

In these cases it is needed to create an intermediary variable, to be mapped in the IEC 60870-5-104 client, and properly convert the types, as can be observed on the example's code below.

```
PROGRAM UserPrg
VAR
  iAnalogIn:    INT;
  iAnalogOut:   INT;
  diCounter:    DINT;
END_VAR

// Analog input conversion from WORD (PROFIBUS) to INT (IEC104)
iAnalogIn:= WORD_TO_INT(wNX6000in00);

// Analog output conversion from INT(IEC104) to WORD (PROFIBUS)
wNX6100out00:= INT_TO_WORD(iAnalogOut);

// Counter conversion from WORDs high+low (PROFIBUS) to DINT (IEC104)
diCounter:= DWORD_TO_DINT(ROL(WORD_TO_DWORD(wNX1005cnt00H), 16) OR wNX1005cnt00L
   );
```

5.5.14.2. Double Points

The double digital points are used to indicate equipment position, such as valves, circuit breakers and secctioners, where the transition between open and close states demand a determined time. Can thus indicate an intermediary transition state between both final states.

Double digital points are also used as outputs and, in an analogous way, it is necessary to keep one of the outputs enabled for a certain time to complete the transition. Such actuation is done through pulses, also known by trip/close commands, with determined duration (enough to the switching of the device under control).

Consult the Double Points section of Utilization Manual for information about double digital points through DBP data type.

Once the Nexto Series digital input and output modules don't support DBP points mapping, some application trickery are needed to make it possible. Remembering that is also not possible to use the *PulsedCommand* function, defined at the *LibRtuStandard* library, to operate the Nexto Series digital double points.

5.5.14.2.1. Digital Input Double Points

For the digital input modules it is needed two auxiliary variables' declaration, to be mapped on the digital input module, besides the double point that is wished to map on the server:

- The double point value variable: type DBP
- The simple point OFF/TRIP value variable: type BOOL
- The simple point ON/CLOSE value variable: type BOOL

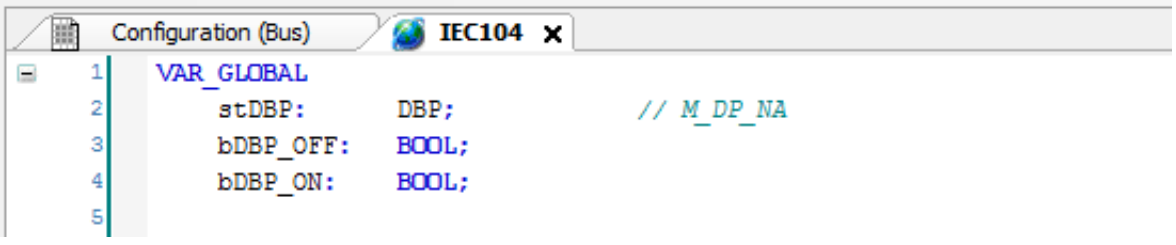


Figure 112: Double Point Variables Declaration Example

Done the variables declaration, it is necessary to create a link between the value variables and the digital input module quality, through the CPU's *Internal Points* tab:

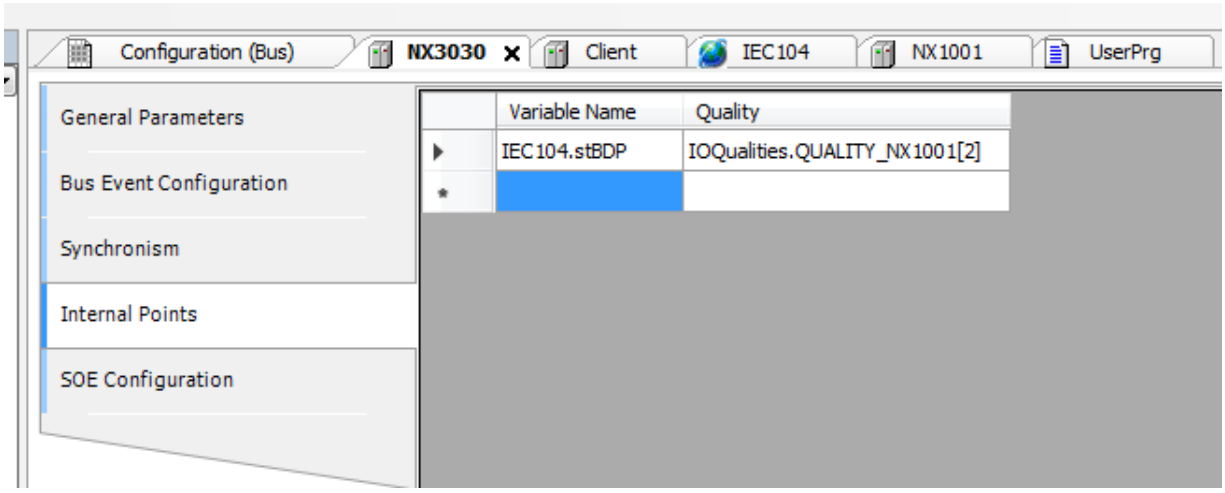


Figure 113: Double Point Variables Attribution to Internal Points

The double point value variable must be mapped at the server IEC 60870-5-104 driver, and both simple variables at the Nexto Series digital input module (in that example, a NX1001). Typically the OFF (TRIP) state is mapped to the even input and the ON (CLOSE) state to the odd input.

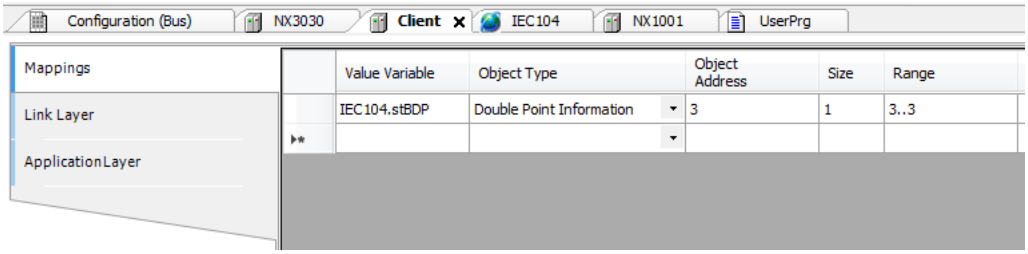


Figure 114: Double Point Variables Mapping on the Client IEC 60870-5-104

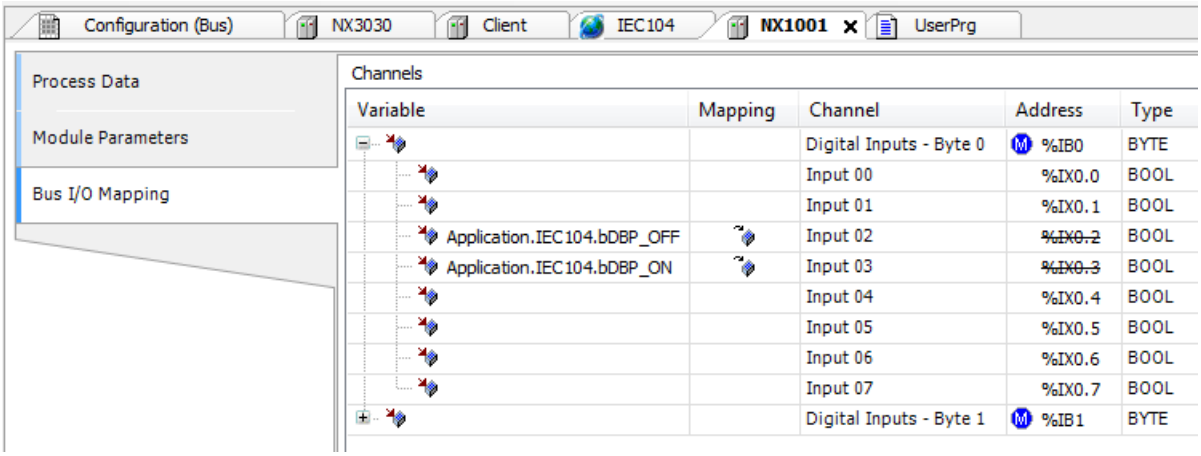


Figure 115: Variables Mapping at the Module Inputs

At last, the user must insert two code lines in its application, to be cyclically executed, to simple variables value attribution to double point:

- DBP value variable, index ON, receive simple point ON value
- DBP value variable, index OFF, receive simple point OFF value

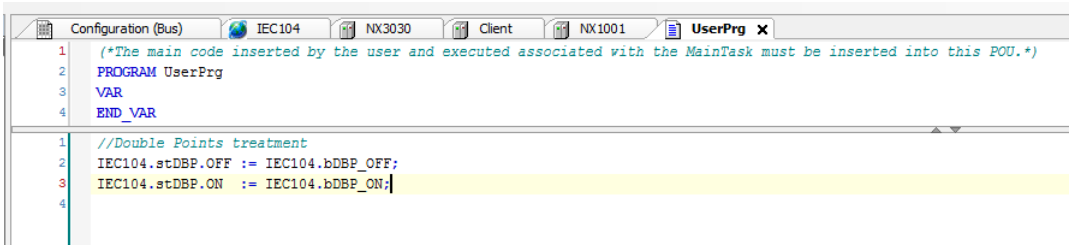


Figure 116: Variables' Values Attribution to the Double Point

5.5.14.2.2. *Digital Output Double Points*

For the digital output modules it must be used the *CommandReceiver* function block to intercept double points actuation commands originated from the clients IEC 60870-5-104. Consult the section Interception of Commands Coming from the Control Center for further information.

The example code below, POU *CmdRcv*, treats pulsed commands received from clients for a digital double point, mapped in a NX2020 module. Besides the following ST code it is need to map a DBP point in Nexto's IEC 60870-5-104 server.

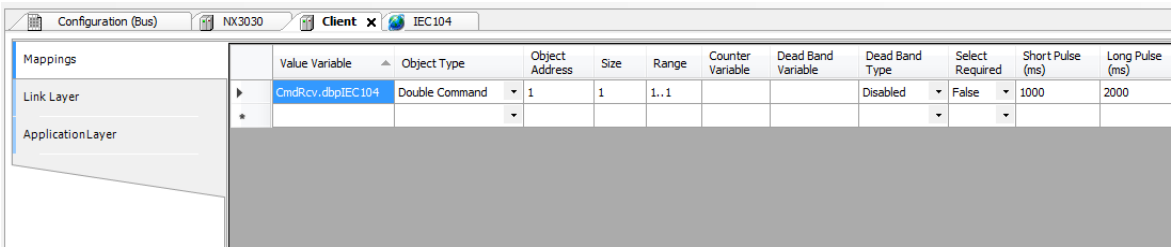


Figure 117: Mapping of Digital Output Double Point variables on IEC 60870-5-104 Client

```
PROGRAM CmdRcv
VAR
CmdReceive:    CommandReceiver; // Interceptor Instance
fbPulsedCmd:  PulsedCommandNexto; // Pulsed Command Instance
byResult:    BYTE; // Pulsed command result
dbpIEC104:     DBP; // Variable mapped in the IEC 104
bSetup:    BOOL:= TRUE; // Interceptor initial setup
END_VAR

// Executes the function configuration in the first cycle
IF bSetup THEN
CmdReceive.dwVariableAddr:= ADR(dbpIEC104);
CmdReceive.bExec:= TRUE;
CmdReceive.eCommandResult:= COMMAND_RESULT.NONE;
CmdReceive.dwTimeout:= 256 * 10;
bSetup:= FALSE;
END_IF

// In case a command is captured:
IF CmdReceive.bCommandAvailable THEN

// Treats each one of the possible commands
```

```
CASE CmdReceive.sCommand.eCommand OF

COMMAND_TYPE.NO_COMMAND:

  // Inform that there is an invalid command.
  // Does nothing and must move on by time-out.

COMMAND_TYPE.SELECT:

  // Treats only commands for double points
  IF CmdReceive.sCommand.sSelectParameters.sValue.eParamType =
   DOUBLE_POINT_COMMAND THEN
    // Returns command finished with success
    // (controlled by IEC104 protocol)
    byResult:= 7;
  ELSE
    // Returns command not supported
    byResult:= 1;
  END_IF

COMMAND_TYPE.OPERATE:

  // Treats only commands for double points
  IF CmdReceive.sCommand.sOperateParameters.sValue.eParamType =
   DOUBLE_POINT_COMMAND THEN
    // Pulse generation in outputs
    IF CmdReceive.sCommand.sOperateParameters.sValue.sDoublePoint.bValue THEN
      // Executes TRIP function
      fbPulsedCmd(
        byCmdType:= 101,
        byPulseTime:= DWORD_TO_BYTE(CmdReceive.sCommand.sOperateParameters.
    sValue.sDoublePoint.sPulseConfig.dwOnDuration/10),
        ptDbpVarAdr:= ADR(dbpIEC104),
        stQuality:= IOQualities.QUALITY_NX2020[4],
        byStatus=> byResult);
    ELSE
      // Executes CLOSE function
      fbPulsedCmd(
        byCmdType:= 102,
        byPulseTime:= DWORD_TO_BYTE(CmdReceive.sCommand.sOperateParameters.
    sValue.sDoublePoint.sPulseConfig.dwOffDuration/10),
        ptDbpVarAdr:= ADR(dbpIEC104),
        stQuality:= IOQualities.QUALITY_NX2020[5],
        byStatus=> byResult);
    END_IF
  ELSE
    // Returns command not supported
    byResult:= 1;
  END_IF

COMMAND_TYPE.CANCEL:

  // Returns command finished with success
  // (controlled by IEC104 protocol)
  byResult:= 7;
```

```
END_CASE

// Treats the pulsed command function result
// and generates the answer to the intercepted command
CASE byResult OF
1:  // Invalid type of command
  CmdReceive.eCommandResult:= COMMAND_RESULT.NOT_SUPPORTED;
  CmdReceive.bDone:= TRUE;
2:  // Invalid input parameters
  CmdReceive.eCommandResult:= COMMAND_RESULT.INCONSISTENT_PARAMETERS;
  CmdReceive.bDone:= TRUE;
3:  // Parameter change in running
  CmdReceive.eCommandResult:= COMMAND_RESULT.PARAMETER_CHANGE_IN_EXECUTION;
  CmdReceive.bDone:= TRUE;
4:  // Module did not answered the command(absent)
  CmdReceive.eCommandResult:= COMMAND_RESULT.HARDWARE_ERROR;
  CmdReceive.bDone:= TRUE;
5:  // Command started and in running (does not returns nothing)
6:  // Another command has been sent to this point and it is running
  CmdReceive.eCommandResult:= COMMAND_RESULT.LOCKED_BY_OTHER_CLIENT;
  CmdReceive.bDone:= TRUE;
7:  // Command finished with success
  CmdReceive.eCommandResult:= COMMAND_RESULT.SUCCESS;
  CmdReceive.bDone:= TRUE;
END_CASE

END_IF

CmdReceive();

IF CmdReceive.bDone THEN
CmdReceive.bDone:= FALSE;
END_IF
```

As can be observed in the previous code, to help in the pulse generation in Nexto's digital double outputs, it was created and used a function block equivalent to *PulsedCommand* function of library *LibRtuStandard*. The *PulsedCommandNexto()* function block shows up coded in ST language.

```
FUNCTION_BLOCK PulsedCommandNexto
VAR_INPUT
byCmdType:    BYTE;     // command type:
            // 100 = status
            // 101 = close/on
            // 102 = trip/off
byPulseTime:  BYTE;     // Pulse duration (in hundredths of second)
ptDbpVarAdr:  POINTER TO DBP; // DBP variable address (can be mapped)
stQuality:    QUALITY;    // DBP point quality(digital module)
END_VAR
VAR_OUTPUT
bON:      BOOL;   // Odd output mapped on Nexto DO module
bOFF:     BOOL;   // Even output mapped on Nexto DO module
byStatus:   BYTE:= 7; // Function return:
          // 1 = invalid command
          // 2 = Time out of valid range (2..255)
```

```
            // 3 = command changed in running time
            // 4 = module did not answer to the command (absent)
            // 5 = command started or running
            // 6 = There is already an active command on this point
            // 7 = pulse command finished with success
END_VAR
VAR
byState:    BYTE;   // Function block state
udiPulseEnd:  UDINT;  // Pulse end instant
END_VAR

// PulsedCommandNexto state machine
CASE byState OF

0: // Init state, ready to receive commands:
CASE byCmdType OF

  100:// Just returns the last status

  101:  // Execute pulse ON:
    // Valids the pulse duration
    IF byPulseTime > 1 THEN
      // Check if there is already an active command on this point
      IF ptDbpVarAdr^.ON OR ptDbpVarAdr^.OFF THEN
        // Returns that there is already an active command
        byStatus:= 6;
      ELSE
        // Enables CLOSE output
        ptDbpVarAdr^.ON:= TRUE;
        ptDbpVarAdr^.OFF:= FALSE;
        // Next state: execute pulse ON
        byState:= byCmdType;
        // Returns started command
        byStatus:= 5;
      END_IF
    ELSE
      // Returns the out of range pulse
      byStatus:= 2;
    END_IF

  102:  // Execute pulse OFF
    // Valids the pulse duration
    IF byPulseTime > 1 THEN
      // Check if there is already an active command on this point
      IF ptDbpVarAdr^.ON OR ptDbpVarAdr^.OFF THEN
        // Returns that there is already an active
        byStatus:= 6;
      ELSE
        // Enables TRIP output
        ptDbpVarAdr^.ON:= FALSE;
        ptDbpVarAdr^.OFF:= TRUE;
        // Next step: execute pulse OFF
        byState:= byCmdType;
        // Returns started command
        byStatus:= 5;
      END_IF
```

```
      ELSE
        // Returns the out of range pulse
        byStatus:= 2;
      END_IF
  ELSE
    // Returns invalid command
    byStatus:= 1;
  END_CASE

  // Memorizes the instant of the pulse end
  udiPulseEnd:= SysTimeGetMs() + BYTE_TO_UDINT(byPulseTime) * 10;

  101, 102:// Continues the pulse execution ON/OFF
  // It returns that the command is running
  byStatus:= 5;
  // Checks the running parameter change
  IF byCmdType <> 100 AND byCmdType <> byState THEN
    // Returns the running parameter change
    byStatus:= 3;
  END_IF
  // Checks pulse end
  IF SysTimeGetMs() >= udiPulseEnd THEN
    // Disable TRIP and CLOSE outputs
    ptDbpVarAdr^.ON:= FALSE;
    ptDbpVarAdr^.OFF:= FALSE;
    // Returns finished command, only if the command has not changed
    IF byCmdType = 100 OR byCmdType = byState THEN
      byStatus:= 7;
    END_IF
    // Next state: initial
    byState:= 0;
  END_IF

  END_CASE

  // Checks digital module (DBP point) quality
  IF stQuality.VALIDITY <> QUALITY_VALIDITY.VALIDITY_GOOD THEN
  // Disable TRIP and CLOSE outputs
  ptDbpVarAdr^.ON:= FALSE;
  ptDbpVarAdr^.OFF:= FALSE;
  // Returns absent module
  byStatus:= 4;
  // Next state: initial
  byState:= 0;
  END_IF

  // Copy DBP output states to the simple outputs
  bON:= ptDbpVarAdr^.ON;
  bOFF:= ptDbpVarAdr^.OFF;
```

5.5.14.3. General Parameters

To the *General Parameters* configuration of an IEC 60870-5-104 Server according to figure below follow the table below parameters:

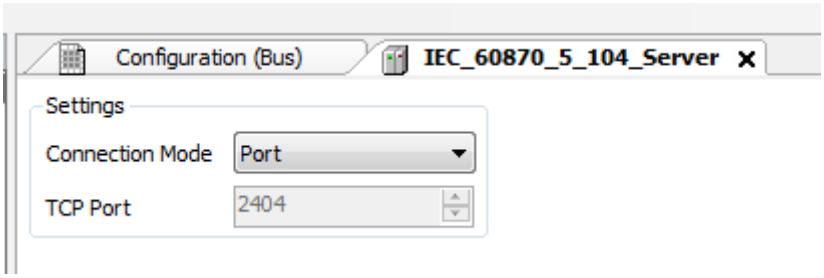


Figure 118: Server IEC 60870-5-104 General Parameters Screen

Parameter	Description	Factory Default	Possibilities
Connection Mode	Set the connection mode with the Connected Client modules.	Port	Port IP
TCP Port	Defines which PLC's TCP port number will be used to communicate with the Connected Client modules. In case the "Connection Mode" field is set as "IP".	2404	1 to 65535

Table 141: IEC 60870-5-104 Server General Parameters Configuration

5.5.14.4. Data Mapping

To configure the IEC 60870-5-104 Server data relation, viewed on figure below follow the parameters described on table below:

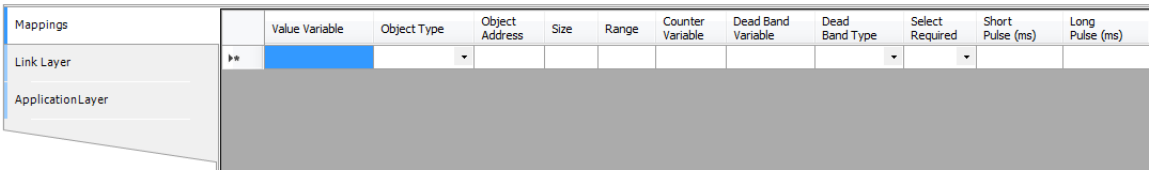


Figure 119: IEC 60870-5-104 Server Mappings Screen

Parameter	Description	Factory Default	Possibilities
Value Variable	Symbolic variable name	-	Name of a variable declared in a POU or GVL
Object Type	IEC 60870-5-104 object type configuration	-	Single Point Information Double Point Information Step Position Information Measured Value (Normalized) Measured Value (Scaled) Measured Value (Short Floating Point) Integrated Totals Bitstring Information (32 Bits) Single Command Double Command Regulating Step Command Setting Point Command (Normalized) Setting Point Command (Scaled) Setting Point Command (Short Floating Point) Bitstring Command (32 Bits)
Object Address	IEC 60870-5-104 mapping first point's index	-	1 to 65535
Size	Specifies the maximum data quantity that an IEC 60870-5-104 mapping will can access	-	1 to 86400000
Range	Configured data address range	-	-
Counter Variable	Name of the symbolic variable which will hold the counter variable's value	-	Name of a variable declared in a POU, GVL or counter module
Dead Band Variable	Name of the symbolic variable which will hold the dead band's value	-	Name of a variable declared in a POU or GVL
Dead Band Type	Defines the dead band type to be used in the mapping	Disabled	Absolute Disabled Integrated
Select Required	Defines if it is required a previous select to run a command	False	True False
Short Pulse (ms)	Defines the short pulse time to an IEC 60870-5-104 digital command	1000	1 to 86400000
Long Pulse (ms)	Defines the long pulse time to an IEC 60870-5-104 digital command	2000	1 to 86400000

Table 142: IEC 60870-5-104 Server Mappings Configuration

Notes:

Value Variable: When a read command is sent, the return received in the answer is stored in this variable. When it is a write command, the written value is going to be stored in that variable. The variable can be simple, array, array element or can be at structures.

Counter Variable: This field applies only on mapping of *Integrated Totals* type objects, being this the controller variable to be managed on process. It must has same type and size of the variable declared on *Value Variable* column, which value is going to be read, or reported to, the client in case of events.

ATTENTION

When the *Counter Variable* has a quality variable associated, to the quality to be transferred to the frozen variable at freeze command, it must be associated a quality variable to the frozen one. This procedure must be done through *Internal Points* tab.

Dead Band Variable: This field applies only to input analog variables (*Measured Value* type objects) mappings. It must has same type and size of the variable declared on *Value Variable* column. New dead band variable values are going to be considered only when the input analog variable change its value.

Dead Band Type: The configuration types available to dead band are:

Function type	Configuration	Description
Dead Band Type	Disabled	In this option, any value change in a group's point, as smaller it is, generates an event to this point.
	Absolute	In this option, if the group's point value absolute change is bigger than the value in "Dead Band" field, an event is going to be generated to this point.
	Integrated	In this option, if the absolute of the integration of the group's point value change is bigger than the value in "Dead Band" field, an event is going to be generated to this point. The integration interval is one second.

Table 143: IEC 60870-5-104 Server Mappings Dead Band Types

Short Pulse and Long Pulse: At the define of short and long pulses duration time it must be considered the limits supported by the device which will treat the command. For example, case the destiny is an output card, which is not supported in native by Nexto Series. It must be checked at the module's Datasheet what the minimum and maximum times, as well as the resolution, to running the pulsed commands.

5.5.14.5. Link Layer

To the IEC 60870-5-104 Server link layer parameters configuration, shown on figure below, follow the described parameters on table below:

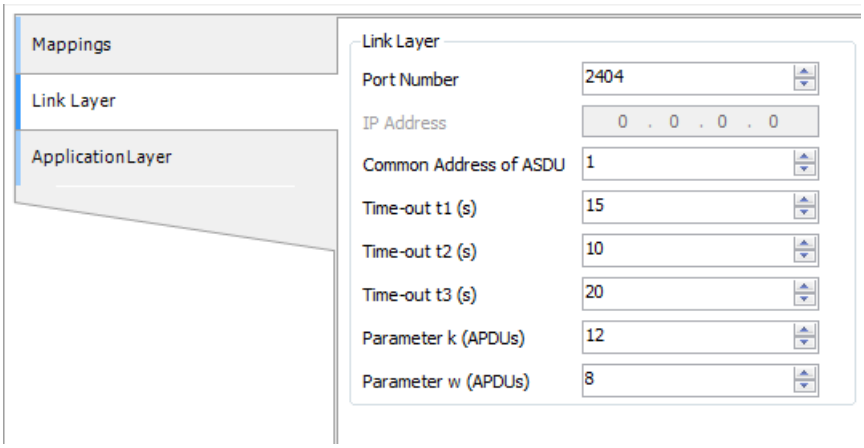


Figure 120: Server IEC 60870-5-104 Link Layer Configuration Screen

Parameter	Description	Factory Default	Possibilities
Port Number	Listened port address to client connection. Used when the client connection isn't through IP	2404	1 to 65535
IP Address	Connected client IP, used when the client connection is through IP	0.0.0.0	1.0.0.1 to 223.255.255.254
Common Address of ASDU	IEC 60870-5-104 address, if the connected client is through IP	1	1 to 65534
Time-out t1 (s)	Time period (in seconds) that the device waits the receiving of an acknowledge message after sent an APDU message type I or U (data), before close the connection	15	1 to 180
Time-out t2 (s)	Time period (in seconds) that the device waits to send a watch message (S-Frame) acknowledging the data frame receiving	10	1 to 180
Time-out t3 (s)	Time period (in seconds) in what is going to be sent a message to link test in case there is no transmission by both sides	20	1 to 180
Parameter k (APDUs)	Maximum number of data messages (I-Frame) transmitted and not acknowledged	12	1 to 12
Parameter w (APDUs)	Maximum number of data messages (I-Frame) received and not acknowledged	8	1 to 8

Table 144: IEC 60870-5-104 Server Link Layer Configuration

Note:

The fields *Time-out t1 (s)*, *Time-out t2 (s)* and *Time-out t3 (s)* are dependents between themselves and must be configured in a way that *Time-out t1 (s)* be bigger than *Time-out t2 (s)* and *Time-out t3 (s)* be bigger than *Time-out t1 (s)*. If any of these rules be not respected, error messages are going to be generated during the project compilation.

ATTENTION

For slow communication links (example: satellite communication), the parameters *Time-out t1 (s)*, *Time-out t2 (s)* and *Time-out t3 (s)* must be properly adjusted, such as doubling the default values of these fields.

5.5.14.6. Application Layer

To configure the IEC 60870-5-104 Server application layer, shown on figure below, follow the parameters described on table below:

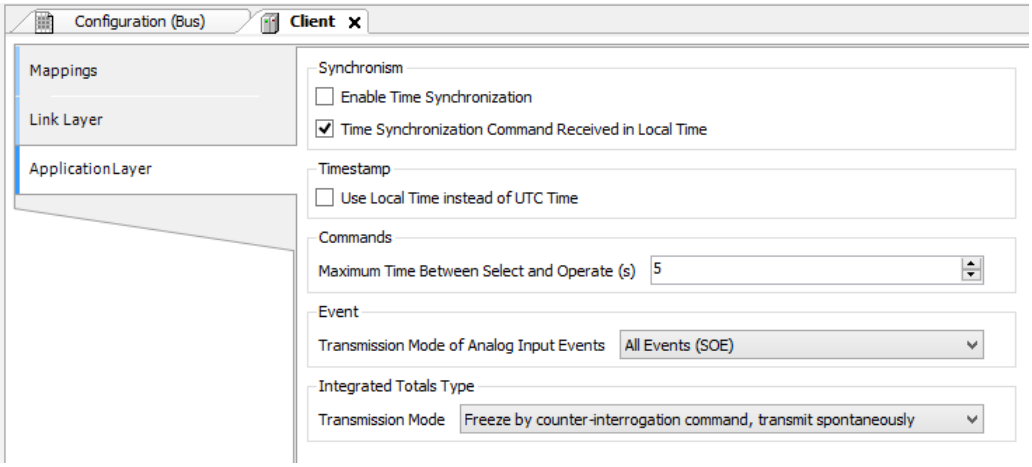


Figure 121: Server IEC 60870-5-104 Application Layer Configuration Screen

Parameter	Description	Factory Default	Possibilities
Enable Time Synchronization	Option to Enable/Disable time sync request	Disabled	Disabled Enabled
Time Synchronization Command Received in Local Time	Option to Enable/Disable the treatment of the synchronization command in local time	Enabled	Disabled Enabled
Use Local Time instead of UTC Time	Option to Enable/Disable the time stamp in local time for events	Disabled	Disabled Enabled
Maximum Time Between Select and Operate (s)	Time period in which the selection command will remain active (the count starts from the received selection command acknowledge) waiting the Operate command	5	1 to 180

Parameter	Description	Factory Default	Possibilities
Transmission Mode of Analog Input Events	Analog input events transmission mode	All Events (SOE)	All Events (SOE) Most Recent Event
Transmission Mode	Frozen counters transmission mode (Integrated Totals)	Freeze by counter-interrogation command, transmit spontaneously	Freeze by counter-interrogation command, transmit spontaneously Freeze and transmit by counter-interrogation command

Table 145: IEC 60870-5-104 Server Application Layer Configuration

Notes:

Enable Time Synchronization: Once enabled, allow the IEC 60870-5-104 Server adjust the CPU's clock when a sync command is received.

Time Synchronization Command Received in Local Time: When enabled, the IEC 60870-5-104 Server adjusts the CPU clock by treating the time received in the synchronization command as local time. Otherwise, this time is considered UTC.

Use Local Time instead of UTC Time: Once enabled, the time stamp of the events generated by IEC 60870-5-104 Server will be sent according to the CPU's local time.

ATTENTION

When the time sync option is checked in more than one server, the received times from different servers will be overwritten in the system clock in a short time period, being able to cause undesirable behaviors due to delays on messages propagation time and system load.

Transmission Mode of Analog Inputs Events: The Analog Inputs Events transmission modes available are the following:

Function Type	Configuration	Description
Transmission Mode of Analog Input Events	All Events (SOE)	All analog events generated are going to be sent.
	Most Recent Event	It is sent only the most recent analog event.

Table 146: IEC 60870-5-104 Server Transmission Modes of Analog Inputs Events

Transmission Mode: The available transmission modes of the frozen counters (*Integrated Totals*) are the following:

Function Type	Configuration	Description
Transmission Mode	Freeze by counter-interrogation command, transmit spontaneously	Equivalent to the counters acquisition D Mode (Integrated Totals) defined by Standard IEC 60870-5-101. In this mode, the control station's counters interrogation commands, freeze the counters. Case the frozen values have been modified, they are reported through events.
	Freeze and transmit by counter-interrogation command	Equivalent to the counters acquisition C Mode (Integrated Totals) defined by Standard IEC 60870-5-101. In this mode, the control station's counters interrogation commands, freeze the counters. The subsequent counters interrogation commands (read) are sent by the control station to receive the frozen values.

Table 147: IEC 60870-5-104 Server Transmission Modes of the Frozen Counters

ATTENTION

The Standard IEC 60870-5-104, section *Transmission control using Start/Stop*, foresee the commands *STARTDT* and *STOPDT* utilization to data traffic control between client and server, using simple or multiple connections. Despite Nexto supports such commands, its utilization isn't recommended to control data transmission, mainly with redundant CPUs, because such commands aren't synchronized between both CPUs. Instead of using multiple connections between client and Nexto server, it's suggested the use of NIC Teaming resources to supply (physically) redundant Ethernet channels and preserve the CPU resources (CPU control centers).

5.5.14.7. Server Diagnostic

The IEC 60870-5-104 Server protocol diagnostics are stored in *T_DIAG_IEC104_SERVER_1* type variables, which are described in table below:

Diagnostic variable of type T_DIAG_IEC104_SERVER_1.*	Size	Description
Command bits, automatically reset:		
tCommand.bStop	BOOL	Disable Driver
tCommand.bStart	BOOL	Enable Driver
tCommand.bDiag_01_Reserved	BOOL	Reserved
tCommand.bDiag_02_Reserved	BOOL	Reserved
tCommand.bDiag_03_Reserved	BOOL	Reserved
tCommand.bDiag_04_Reserved	BOOL	Reserved
tCommand.bDiag_05_Reserved	BOOL	Reserved
tCommand.bDiag_06_Reserved	BOOL	Reserved
Diagnostics:		
tClient_X.bRunning	BOOL	IEC 60870-5-104 Server is running

Diagnostic variable of type T_DIAG_IEC104_SERVER_1.*	Size	Description
tClient_X.eConnectionStatus. CLOSED	ENUM(BYTE)	Communication channel closed. Server won't accept connection request. ENUM value (0)
tClient_X.eConnectionStatus. LISTENING		Server is listening to the configured port and there is no connected clients. ENUM value (1)
tClient_X.eConnectionStatus. CONNECTED		Connected client. ENUM value (2)
tClient_X.tQueueDiags. bOverflow	BOOL	Client queue is overflowed
tClient_X.tQueueDiags. wSize	WORD	Configured queue size
tClient_X.tQueueDiags. wUsage	WORD	Events number in the queue
tClient_X.tQueueDiags. dwReserved_0	DWORD	Reserved
tClient_X.tQueueDiags. dwReserved_1	DWORD	Reserved
tClient_X.tStats.wRXFrames	WORD	Number of received frames
tClient_X.tStats.wTXFrames	WORD	Number of sent frames
tClient_X.tStats.wCommErrors	WORD	Communication errors counter, including physical layer, link layer and transport layer errors.
tClient_X.tStats.dwReserved_0	DWORD	Reserved
tClient_X.tStats.dwReserved_1	DWORD	Reserved

Table 148: IEC 60870-5-104 Server Diagnostics

5.5.14.8. Commands Qualifier

The standard IEC 60870-5-104 foresee four different command qualifiers for the objects *Single Command*, *Double Command* and *Regulating Step Command*, all supported by the Nexto Server.

Each object type has a specific behavior to each command qualifier, as can be seen on the table below.

Qualifier	Protocol IEC 60870-5-104 object type		
	Single Command	**Double Command**	**Regulating Step Command**
No additional definition (default)	Same behavior of persistent qualifier.	Same behavior of short pulse qualifier.	Same behavior of short pulse qualifier.
Short pulse duration	Requires command interception to application treatment. Other way it will return a negative acknowledge message (fail).	Requires command interception to application treatment. Other way it will return a negative acknowledge message (fail).	Requires command interception to application treatment. Other way it will return a negative acknowledge message (fail).
Long pulse duration			

Qualifier	Protocol IEC 60870-5-104 object type		
	Single Command	**Double Command**	**Regulating Step Command**
Persistent output	The output is going to be on or off and that will remain until new command, according to value (ON or OFF) commanded by the client.		

Table 149: IEC 60870-5-104 Server Commands Qualifier

Note:

Command Interception: For further information about commands interception of IEC 608705-104 clients, consult section Interception of Commands Coming from the Control Center, implemented through *CommandReceiver* function block.

5.5.15. PROFINET Controller

For correct use of the PROFINET Controller protocol, it is necessary to consult the manual MU214621 - Nexto Series PROFINET Manual .

5.6. Communication Performance

5.6.1. MODBUS Server

The MODBUS devices configurable in the Nexto CPU run in the background, with a priority below the user application and cyclically. Thus, their performance varies depending on the remaining time, taking into account the difference between the interval and time that the application takes to run. For example, a MODBUS device in an application that runs every 100 ms, with a running time of 50 ms, will have a lower performance than an application running every 50 ms to 200 ms of interval. It happens because in the latter case, the CPU will have a longer time between each MainTask cycle to perform the tasks with lower priority.

It also has to be taken into account the number of cycles that the device, slave or server takes to respond to a request. To process and transmit a response, a MODBUS RTU Slave will takes two cycles (cycle time of the MODBUS task), where as a MODBUS Ethernet Server task takes only one cycle. But this is the minimum time between receipt of a request and the reply. If the request is sent immediately after the execution of a task MODBUS cycle time may be equal to 2 or 3 times the cycle time for the MODBUS slave and from 1 to 2 times the cycle time for the MODBUS server.

In this case: Maximum Response Time = 3 * (cycle time) + (time of execution of the tasks) + (time interframe chars) + (send delay).

For example, for a MODBUS Ethernet Server task with a cycle of 50 ms, an application that runs for 60 ms every 100 ms, the server is able to run only one cycle between each cycle of the application. On the other hand, using the same application, running for 60 ms, but with an interval of 500 ms, the MODBUS performs better, because while the application is not running, it will be running every 50 ms and only each cycle of MainTask it will take longer to run. For these cases, the worst performance will be the sum of the Execution Time of the user application with the cycle time of the MODBUS task.

For the master and client devices the operating principle is exactly the same, but taking into account the polling time of the MODBUS relation and not the cycle time of the MODBUS task. For these cases, the worst performance of a relationship will be performed after the polling time, along with the user application Execution Time.

It is important to stress that the running MODBUS devices number also changes its performance. In an user application with Execution Time of 60 ms and interval of 100 ms, there are 40 ms left for the CPU to perform all tasks of lower priority. Therefore, a CPU with only one MODBUS Ethernet Server will have a higher performance than a CPU that uses four of these devices.

5.6.1.1. CPU's Local Interfaces

For a device MODBUS Ethernet Server, we can assert that the device is capable to answer a x number of requisitions per second. Or, in other words, the Server is able to transfer n bytes per second, depending on the size of each requisition. As smaller is the cycle time of the MODBUS Server task, higher is the impact of the number of connections in his answer rate. However, for cycle times smaller than 20 ms this impact is not linear and the table below must be viewed for information.

The table below exemplifies the number of requisitions that a MODBUS Server inserted in a local Ethernet interface is capable to answer, according to the cycle time configured for the MODBUS task and the number of active connections:

Number of Active Connections	Answered requisitions per second with the MODBUS task cycle at 5 ms	Answered requisitions per second with the MODBUS task cycle at 10 ms	Answered requisitions per second with the MODBUS task cycle at 20 ms
1 Connection	185	99	50
2 Connections	367	197	100
4 Connections	760	395	200
7 Connections	1354	695	350
10 Connections	1933	976	500

Table 150: Communication Rate of a MODBUS Server at Local Interface

ATTENTION

The communication performances mentioned in this section are just examples, using a CPU with only one device MODBUS TCP Server, with no logic to be executed inside the application that could delay the communication. Therefore, these performances must be taken as the maximum rates.

For cycle times equal or greater than 20 ms, the increase of the answer rate is linear, and may be calculated using an equation:

N = C x (1 / T)

Where:

N is the medium number of answers per second;

C is the number of active connections;

T is the MODBUS task interval in seconds.

As an example a MODBUS Server, with only one active connection and a cycle time of 50 ms we get:

C = 1; T = 0,05 s;

N = 1 x (1 / (0,05))

N = 20

That is, in this configuration the MODBUS Server answers, on average, 20 requisitions per second.

In case the obtained value is multiplied by the number of bytes in each requisition, we will obtain a transfer rate of n bytes per second.

5.6.1.2. Remote Interfaces

The performance of a device MODBUS Server in one remote Ethernet interface is similar to the performance in the CPU's local interfaces.

However, due to time of the communication between the CPU and the modules, the maximum performance is limited. For only one active connection, the number of answers is limited in the maximum of 18 answers per second. With more active connections, the number of answers will increase linearly, exactly like the local interfaces, however being limited at the maximum of 90 answers per second. So, for a remote Ethernet interface, we will have the following forms to calculate his performance:

For T $\leq$ 55 ms is used:

N = C x (18.18 – (18.18 / (0.055 x 1000)))

And for T $\geq$ 55 ms is used:

N = C x (Z – (Z / (T x 1000)))

Where N is the medium number of answers per second, C is the number of active connections and T is equal to the cycle time of the MODBUS task (in seconds).

The user must pay attention to the fact that the maximum performance of a device MODBUS Server in one remote Ethernet interface is 90 answers of requisitions per second.

5.6.2. OPC DA Server

Communication performance with OPC DA Server was tested by creating POUs with 1,000 INT variables each. All scenarios were tested with *Single* profile and MainTask Interval at 100 ms. The communication was enabled by the *attribute 'symbol' := 'readwrite'*, to make the data available to the OPC DA Server. The measurements were made while MasterTool was disconnected from the CPU, and MainTask duration was made to last 5%, 50% and 80% of the configured Interval, as seen in table below.

At the OPC Client's side, a SCADA system driver was used. Configured update time was 50 ms. Performance results in these conditions are described in table below.

Total quantity of variables in the PLC's project	**Variable update time at OPC DA Client**		
	5% of CPU Busy	**50% of CPU Busy**	**80% of CPU Busy**
1000	600 ms	800 ms	1400 ms
2000	800 ms	900 ms	2800 ms
5000	1000 ms	2000 ms	6500 ms
10000	2000 ms	4000 ms	13700 ms
15000	3200 ms	6400 ms	20000 ms
20000	4000 ms	8100 ms	25000 ms

Table 151: Communication Rate of an OPC DA Server

5.6.3. OPC UA Server

The OPC UA Server MU214609 analyzes the performance of OPC UA communication in greater detail, including addressing the consumption of Ethernet communication bandwidth. This manual also discusses concepts about the operation of the OPC UA protocol.

5.6.4. IEC60870-5-104 Server

The IEC 60870-5-104 Server driver is executed by the CPU in the same way as the other communication drivers Servers, that is, in the background, with a priority below the user application and cyclically. The task of this The driver specifically executes every 50 ms, and 1 driver execution cycle is enough to process and respond to requests. However, as it is a low priority task, it is not guaranteed to be able to run at this frequency because depends on the percentage of free CPU (difference between the MainTask interval and the time that the user application takes to be executed) and also concurrency with tasks from other protocols configured in the CPU.

To help in the comprehension of the driver IEC 60870-5-104 Server performance are presented the result of some test done with an IEC 60870-5-104 Client simulator, connected to a NX3030 running an IEC 60870-5-104 Server. The configured data base was compose of 900 digital points and 100 analog points (all with quality and time stamp), and the MainTask was using 70 ms (of the 100 ms interval).

- Time to complete a general interrogation command: less than one second
- Time to transfer 900 digital events + 100 analog events: 6 seconds

5.7. System Performance

In cases where the application has only one MainTask user task responsible for the execution of a single Program type programming unit called MainPrg (as in Single Profile), the PLC consumes a certain amount of time for the task to be processed. At that time we call it as *Execution Time*.

In an application the average application *Execution Time* can be known using the MasterTool IEC XE in the *Device* item of its *Devices Tree* as follows:

PLC Logic-> Application-> Task Configuration in the *Monitor* tab, *Average Cycle Time* column.

The user must pay attention to the *Cycle Time* so that it does not exceed 80% of the interval set in the MainTask user task. For example, in an application where the interval is 100 ms, an appropriate *Cycle Time* is up to 80 ms. This is due to the fact that the CPU needs time to perform other tasks such as communication processing, processing of the display and memory card, and these tasks take place within the range (the remaining 20% of *Cycle Time*).

ATTENTION

For very high cycle times (typically higher than 300 ms), even that the value of 80% is respected, it may occur a reduction in the display response time and of the diagnostics key. In case the 80 percentage is not respected and the running time of the user task is closer or exceeds the interval set for the MainTask, the screen and the diagnosis button cannot respond once its priority in the system running is lower than the user tasks. In case an application with errors is loaded in the CPU, it may be necessary to restart it without loading this application as shown in the System Log section.

ATTENTION

The CPU's system logs of the Nexto Series, starting from firmware version 1.4.0.33 now reloaded in case of a CPU reset or a reboot of the *Runtime System*, that is, you can view the older logs when one of these conditions occurs.

5.7.1. I/O Scan Time

For a project that uses digital I/O modules, being them inserted into the bus and declared in the project, the MainTask time will increase according to the number of modules. The table below illustrates the average time that is added to the MainTask:

Declared Modules in the Bus	Added Time in the MainTask Cycle Time (μs)
5	300
10	700
20	1000

Table 152: I/O Scanning Time

In projects that use remote I/Os, for example, using the NX5001 PROFIBUS-DP Master module, the manual of the respective module has to be consulted for information about performance and influences of the module in the execution of user tasks.

5.7.2. Memory Card

Data transfers involving the memory card is performed by the CPU in the background, as this gives priority to the execution of user application and communication processing. Thus, the transfer of files to the card may suffer an additional significant time, depending on the Cycle Time of the user application.

The time required to read/write files on the card will be directly affected by the Cycle Time of the user application since this application has priority in execution.

Further information about the use of the memory card see Memory Card section.

5.8. RTC Clock

The CPUs have an internal clock that can be used through the *NextoStandard.lib* library. This library is automatically loaded during the creation of a new project (to perform the library insertion procedure, see Libraries section). The figure below shows how to include the blocks in the project:

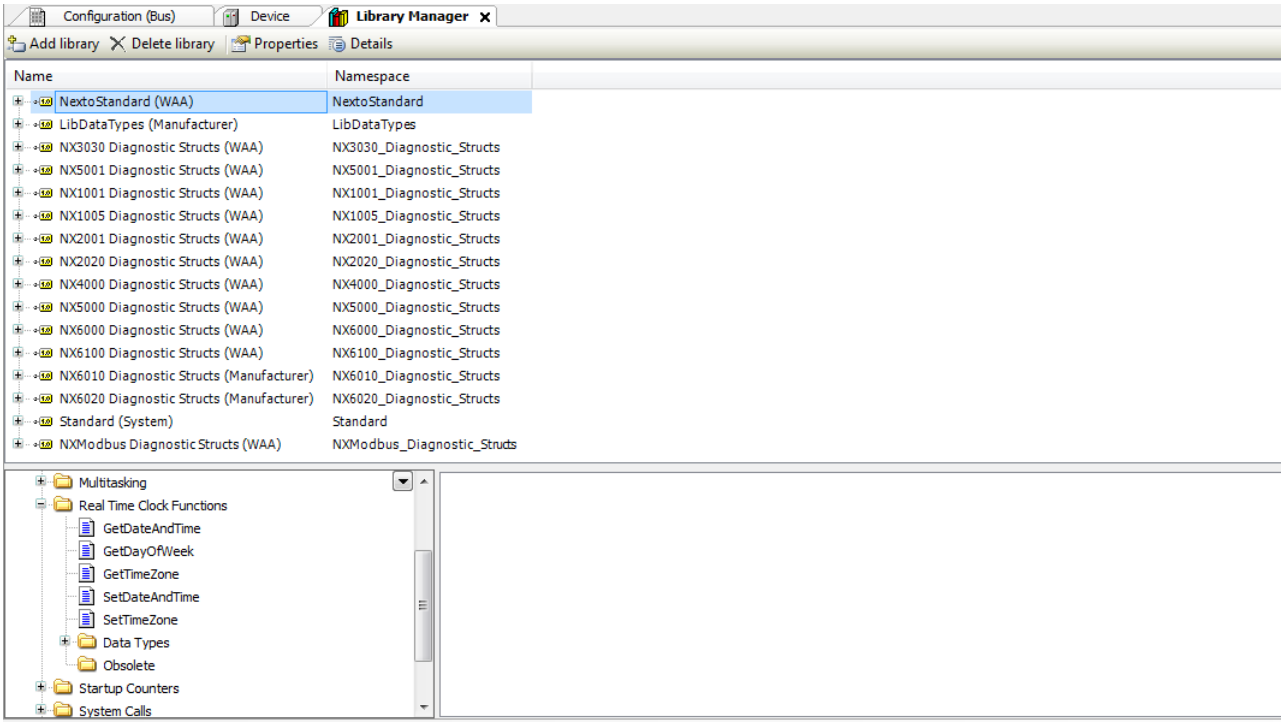


Figure 122: Clock Reading and Writing Blocks

ATTENTION

Function blocks of RTC Reading and Writing, previously available in 2.00 MasterTool IEC XE or older become obsolete from 2.00 or newer, the following blocks are no longer used: *NextoGetDateAndTime*, *NextoGetDateAndTimeMs*, *NextoGetTimeZone*, *NextoSetDateAndTime*, *NextoSetDateAndTimeMs* and *NextoSetTimeZone*.

5.8.1. Function Blocks for RTC Reading and Writing

Among other function blocks, there are some very important used for clock reading (*GetDateAndTime*, *GetDayOfWeek* and *GetTimeZone*) and for date and time new data configuring (*SetDateAndTime* and *SetTimeZone*). These functions always use the local time, that is, take into account the value defined by the *Time Zone*.

The proceedings to configure these two blocks are described below.

ATTENTION

The obsolete function blocks for reading and writing the RTC (*NextoGetDateAndTime*, *NextoGetDateAndTimeMs*, *NextoSetDateAndTime* and *NextoSetDateAndTimeMs*) cannot be used in the redundant data area in redundant projects . They must be used in non-redundant POUs, such as the *NonSkippedPrg* POU. More details on how the POU *NonSkippedPrg* works can be found in NonSkippedPrg Program.

5.8.1.1. Function Blocks for RTC Reading

The clock reading can be made through the following functions:

5.8.1.1.1. GetDateAndTime

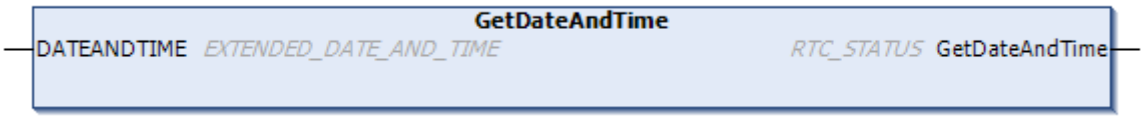


Figure 123: Date and Hour Reading

Input Parameters	Type	Description
DATEANDTIME	EXTENDED_DATE _AND_TIME	This variable returns the value of date and hour of RTC in the format shown at Table 162.

Table 153: Input Parameters of GetDateAndTime

Output Parameters	Type	Description
GETDATEANDTIME	RTC_STATUS	Returns the function error state, see Table 164.

Table 154: Output Parameters of GetDateAndTime

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
Result : RTC_STATUS;
DATEANDTIME : EXTENDED_DATE_AND_TIME;
xEnable : BOOL;
END_VAR
-------------------------------------------------------------------------
IF xEnable = TRUE THEN
Result := GetDateAndTime(DATEANDTIME);
xEnable := FALSE;
END_IF
```

5.8.1.1.2. GetTimeZone

The following function reads the Time Zone configuration, this function is directly related with time in Time Zone at SNTP synchronism service:

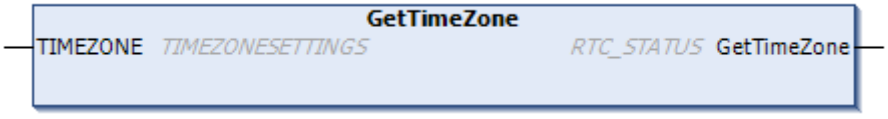


Figure 124: Configuration Reading of Time Zone

Input Parameters	Type	Description
TIMEZONE	TIMEZONESETTINGS	This variable presents the reading of Time Zone configuration.

Table 155: Input Parameters of GetTimeZone

Output Parameters	Type	Description
GetTimeZone	RTC_STATUS	Returns the function error state, see Table 164.

Table 156: Output Parameters of GetTimeZone

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
GetTimeZone_Status   : RTC_STATUS;
TimeZone       : TIMEZONESETTINGS;
xEnable : BOOL;
END_VAR
-------------------------------------------------------------------------
IF xEnable = TRUE THEN
GetTimeZone_Status := GetTimeZone(TimeZone);
xEnable := FALSE;
END_IF
```

5.8.1.1.3. *GetDayOfWeek*

GetDayOfWeek function is used to read the day of the week.

Figure 125: Day of Week Reading

Output Parameters	Type	Description
GetDayOfWeek	DAYS_OF_WEEK	Returns the day of the week. See Section 163.

Table 157: Output Parameters of GetDayOfWeek

When called, the function will read the day of the week and fill the structure *DAYS_OF_WEEK*.
Utilization example in ST language:

```
PROGRAM UserPrg
VAR
DayOfWeek : DAYS_OF_WEEK;
END_VAR
-------------------------------------------------------------------------
DayOfWeek := GetDayOfWeek();
```

5.8.1.2. RTC Writing Functions

The clock settings are made through function and function blocks as follows:

5.8.1.2.1. SetDateAndTime

SetDateAndTime function is used to write the settings on the clock. Typically the precision is on the order of hundreds of milliseconds.

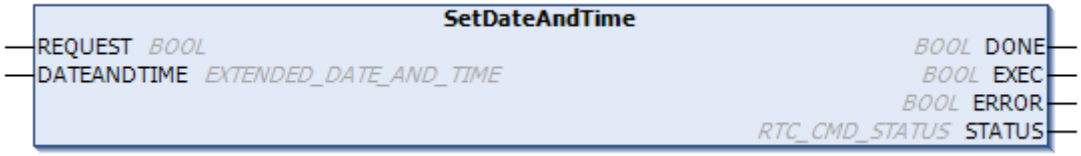


Figure 126: Set Date And Time

Input parameters	Type	Description
REQUEST	BOOL	This variable, when receives a rising edge, enables the clock writing.
DATEANDTIME	EXTENDED_DATE_AND_TIME	Receives the values of date and hour with milliseconds. See section 162.

Table 158: Input Parameters of SetDateAndTime

Output parameters	Type	Description
DONE	BOOL	This variable, when true, indicates that the action was successfully completed.
EXEC	BOOL	This variable, when true, indicates that the function is processing the values.
ERROR	BOOL	This variable, when true, indicates an error during the Writing.
STATUS	RTC_CMD_STATUS	Returns the error occurred during the configuration. See Table 164.

Table 159: Output Parameters of SetDateAndTime

When a rising edge occurs at the *REQUEST* input, the function block will write the new *DATEANDTIME* values on the clock. If the writing is successfully done, the *DONE* output will be equal to *TRUE*. Otherwise, the *ERROR* output will be equal to *TRUE* and the error will appear in the *STATUS* variable.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
SetDateAndTime : SetDateAndTime;
xRequest : BOOL;
DateAndTime : EXTENDED_DATE_AND_TIME;
xDone : BOOL;
xExec : BOOL;
xError : BOOL;
xStatus : RTC_STATUS;
END_VAR
-------------------------------------------------------------------------
IF xRequest THEN
  SetDateAndTime.REQUEST:=TRUE;
  SetDateAndTime.DATEANDTIME:=DateAndTime;
  xRequest:= FALSE;
END_IF
SetDateAndTime();
SetDateAndTime.REQUEST:=FALSE;
IF SetDateAndTime.DONE THEN
  xExec:=SetDateAndTime.EXEC;
  xError:=SetDateAndTime.ERROR;
  xStatus:=SetDateAndTime.STATUS;
END_IF
```

ATTENTION

If you try to write time values outside the range of the RTC, the values are converted to valid values, provided they do not exceed the valid range of 01/01/2000 to 12/31/2035. For example, if the user attempts to write the value 2000 ms, it will be converted to 2 seconds, write the value 100 seconds, it will be converted to 1 min and 40 seconds. If the type value of 30 hours, it is converted to 1 day and 6 hours, and so on.

5.8.1.2.2. SetTimeZone

The following function block makes the writing of the time zone settings:

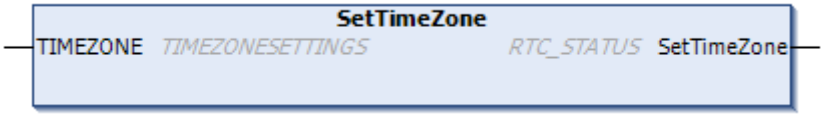


Figure 127: Writing of the Time zone Settings

Input parameters	Type	Description
TIMEZONE	TIMEZONESETTINGS	Structure with time zone to be configured. See Table 165.

Table 160: SetTimeZone Input Parameters

Output parameters	Type	Description
SetTimeZone	RTC_STATUS	Returns the error occurred during the reading/setting. See Table 164.

Table 161: SetTimeZone Output Parameters

When called, the function will configure the *TIMEZONE* with the new system time zone configuration. The configuration results is returned by the function.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
Status : RTC_STATUS;
TimeZone : TIMEZONESETTINGS;
xWrite : BOOL;
END_VAR
--------------------------------------------------------------------------------
//FB SetTimeZone
IF (xWrite = TRUE) THEN
Status := SetTimeZone(TimeZone);
  IF Status = RTC_STATUS.NO_ERROR THEN
    xWrite := FALSE;
  END_IF
END_IF
```

ATTENTION

To perform the clock should be used time and date values within the following valid range: 00:00:00 hours of 01/01/2000 to 12/31/2035 23:59:59 hours, otherwise , is reported an error through the *STATUS* output parameter. For details of the *STATUS* output parameter, see the section RTC_STATUS.

5.8.2. RTC Data Structures

The reading and setting function blocks of the Nexto Series CPUs RTC use the following data structures in its configuration:

5.8.2.1. EXTENDED_DATE_AND_TIME

This structure is used to store the RTC date when used the function blocks for date reading/setting within milliseconds of accuracy. It is described in the table below:

Structure	Type	Variable	Description
EXTENDED_DATE_AND_TIME	BYTE	byDayOfMonth	Stores the day of the set date.
	BYTE	ByMonth	Stores the month of the set date.
	WORD	wYear	Stores the year of the set date.
	BYTE	byHours	Stores the hour of the set date.
	BYTE	byMinutes	Stores the minutes of the set date.
	BYTE	bySeconds	Stores the seconds of the set date.
	WORD	wMilliseconds	Stores the milliseconds of the set date.

Table 162: EXTENDED_DATE_AND_TIME

5.8.2.2. DAYS_OF_WEEK

This structure is used to store the day of week:

Enumerable	Value	Description
DAYS_OF_WEEK	0	INVALID_DAY
	1	SUNDAY
	2	MONDAY
	3	TUESDAY
	4	WEDNESDAY
	5	THURSDAY
	6	FRIDAY
	7	SATURDAY

Table 163: DAYS_OF_WEEK Structure

5.8.2.3. RTC_STATUS

This enumerator is used to return the type of error in the RTC setting or reading and it is described in the table below:

Enumerator	Value	Description
RTC_STATUS	NO_ERROR (0)	There is no error.
	UNKNOWN_COMMAND (1)	Unknown command.
	DEVICE_BUSY (2)	Device is busy.
	DEVICE_ERROR (3)	Device with error.
	ERROR_READING_OSF (4)	Error in the reading of the valid date and hour flag.
	ERROR_READING_RTC (5)	Error in the date and hour reading.
	ERROR_WRITING_RTC (6)	Error in the date and hour writing.
	ERROR_UPDATING_SYSTEM_TIME (7)	Error in the update of the system's date and hour.
	INTERNAL_ERROR (8)	Internal error.
	INVALID_TIME (9)	Invalid date and hour.
	INPUT_OUT_OF_RANGE (10)	Out of the limit of valid date and hour for the system.

Enumerator	Value	Description
	SNTP_NOT_ENABLE (11)	Error generated when the SNTP service is not enabled and it is done an attempt for modifying the time zone.

Table 164: RTC_STATUS

5.8.2.4. TIMEZONESETTINGS

This structure is used to store the time zone value in the reading/setting requests of the RTC's function blocks and it is described in table below:

Structure	Type	Variable	Description
TIMEZONESETTINGS	INT	iHour	Set time zone hour.
	INT	iMinutes	Set time zone minute.

Table 165: TIMEZONESETTINGS

Note:

Function Blocks of Writing and Reading of Date and Hour: different libraries of *NextoStandard*, which have function blocks or functions that may perform access of reading and writing of date and hour in the system, are not indicated. The *NextoStandard* library has the appropriate interfaces for writing and reading the system's date and hour accordingly and for informing the correct diagnostics.

5.9. User Files Memory

Nexto Series CPUs have a memory area destined to the general data storage, in other words, the user can store several project files of any format in the CPU memory. This memory area varies according to the CPU model used (check Memory section).

In order to use this area, the user must access a project in the MasterTool IEC XE software and click on the *Devices Tree*, placed at the program left. Double click on the *Device* item and, after selecting the CPU in the *Communication Settings* tab which will be open, select the *Files* tab and click on *Refresh*, both in the computer files column (left) and in the CPU files column (right) as shown on figure below.

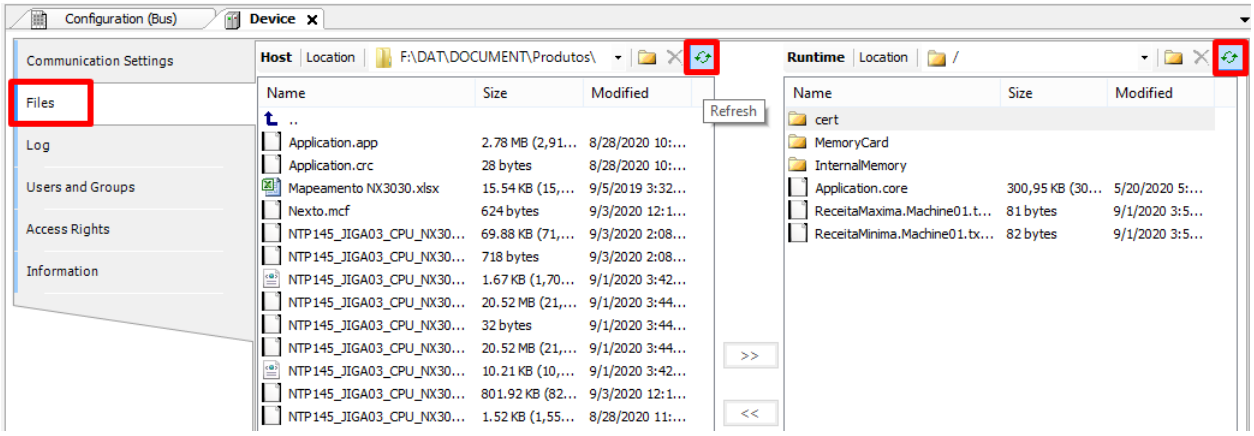


Figure 128: User Files Access

After updating the CPU column of files, the root directory of files stored in the CPU will be shown. Then it will be possible to select the folder where the files will be transferred to. The *"InternalMemory"* folder is a default folder to be used to store

files in the CPU's internal memory, since it is not possible to transfer files to the root directory. If necessary, the user can create other folders in the root directory or subfolders inside the *"InternalMemory"* folder.

The *"MemoryCard"* folder is the directory where the memory card is mounted, if it is inserted into the CPU. Files which are transferred to the *"MemoryCard"* are being transferred directly into the memory card. As new features are being added to the product, some folders may appear and which should be ignored by the user.

ATTENTION

In the case where the memory card is inserted after the CPU startup, an username and password will be requested to perform the MasterTool IEC XE access and/or file transfers to the memory card or vice versa. The standard user with privileges to access the CPU is *"Owner"* and the default password for that user is *"Owner"*.

In order to perform a file transfer from the microcomputer to the CPU just select the desired file in the left column and press the "»" key located in the center of the screen, as shown in figure below. The download time will vary depending on file size and cycle time (execution) of the current application of the CPU and may take several minutes.

The user does not need to be in *Run* Mode or connected to the CPU to perform the transfers, since it has the ability to connect automatically when the user performs the transfer.

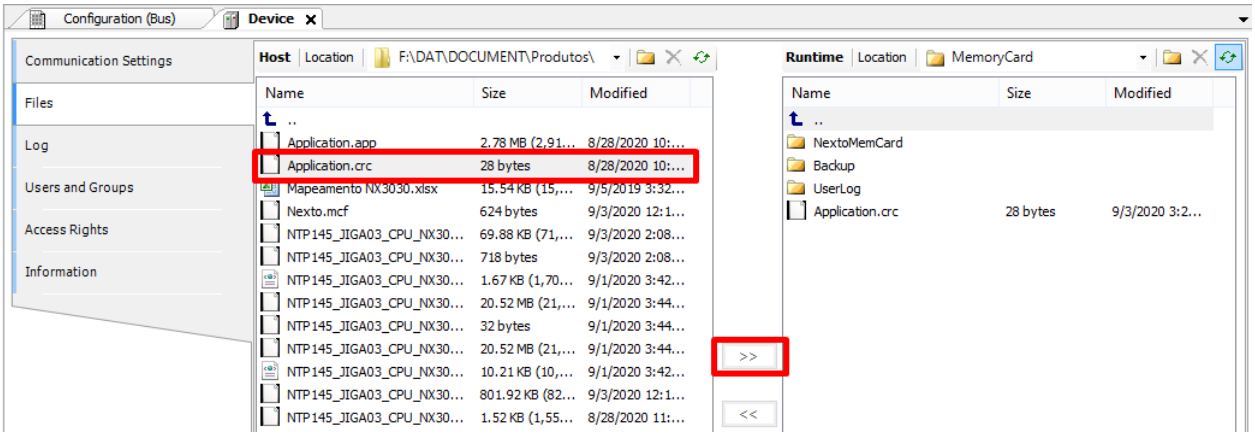


Figure 129: Files Transference

ATTENTION

The files contained in the folder of a project created by MasterTool IEC XE have special names reserved by the system in this way cannot be transferred through the *Files* tab. If the user wishes to transfer a project to the user memory, you must compact the folder and then download the compressed file (*.zip for example).

In case it is necessary to transfer documents from the CPU to the PC in which the MasterTool IEC XE software is installed, the user must follow a very similar procedure to the previously described, as the file must be selected from the right column and the button "«" pressed, placed on the center of the screen.

Furthermore, the user has some operation options in the storing files area, which are the following:

- New directory : allows the creation of a new folder in the user memory area.
- Delete item : allows the files excluding in the folders in the user memory area.
- Refresh : allows the file updating, on the MasterTool IEC XE screen, of the files in the user memory area and in the computer.

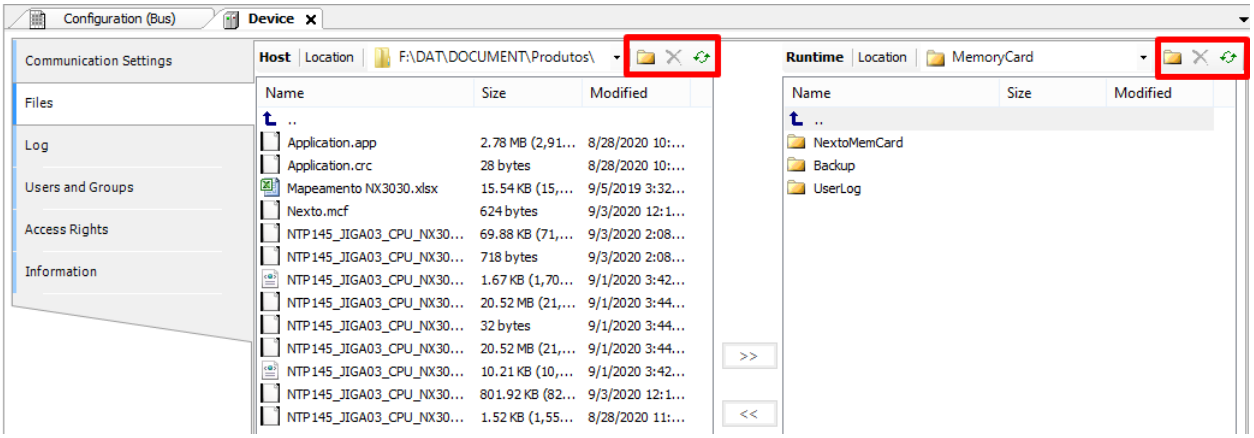


Figure 130: Utilization Options

ATTENTION

For a CPU in Stop Mode or with no application, the transfer rate to the internal memory is approximately 150 Kbytes/s.

5.10. Memory Card

Among other memories, Nexto Series CPUs allow the user the utilization of a memory card. It is defined according the features described in Memory Card Interface section which stores, among other files, the project and application in the CPU internal memory.

When the card is inserted in the CPU and it presents a file type different from FAT32, it automatically identifies those files and questions the user if he wants to format the files. In negative case the user cannot use the card, as it is not mounted. A message informing the format is not recognized is presented and the card presence is not displayed either. If the user decides to format the files, the CPU takes a few minutes to execute the operation, depending on the cycle time (execution) of the application which is running in the CPU. Once the memory card is mounted, the CPU will read its general information, leaving access to the memory card slower in the first few minutes. This procedure is done only when the card is inserted or in case of the CPU reset.

ATTENTION

It is recommended to format the memory card directly in the Nexto CPU in order to avoid possible use problems, mounting time increase or even the incorrect functioning.
It is not recommended to remove the memory card or de-energize the CPU during the formatting or during the files transfer as it can cause the loss of data as well as irreversible damages.

5.10.1. Project Preparation

To use the functionality, during the project configuration, in the MasterTool IEC XE software, the user must enable the option to copy the CPU project to the memory card and/or memory card to the CPU and to configure passwords. These passwords will be requested by the CPU when executing the respective transfer. For information about the table, see section Project Parameters.

ATTENTION

If the CPU has no application, the "*Memory Card*" Menu will be available to allow the transfer of the project from memory card to the CPU without requiring any kind of previous preparation of the CPU.

To use the feature you must perform the following steps.

Navigate to the *Online* menu and execute the command *Create Boot Application*, remembering that you cannot be logged into the CPU to perform this procedure. After you run this command, two files are created in the folder where the project is saved. One with the extension "*app*" and one with the extension "*crc*".

After generating the files in the previous item, you must navigate to the CPU *General Parameters* settings and click the *Memory Card...* button. A new screen will open as shown in figure below. In this screen you must enable the desired transfer operation(s) and, if necessary, set the password(s) with numeric characters only. The use of password is not required.

To complete the setup operation you must click the *Find File...* button and then locate the file with "*.crc*" extension generated in the previous step.

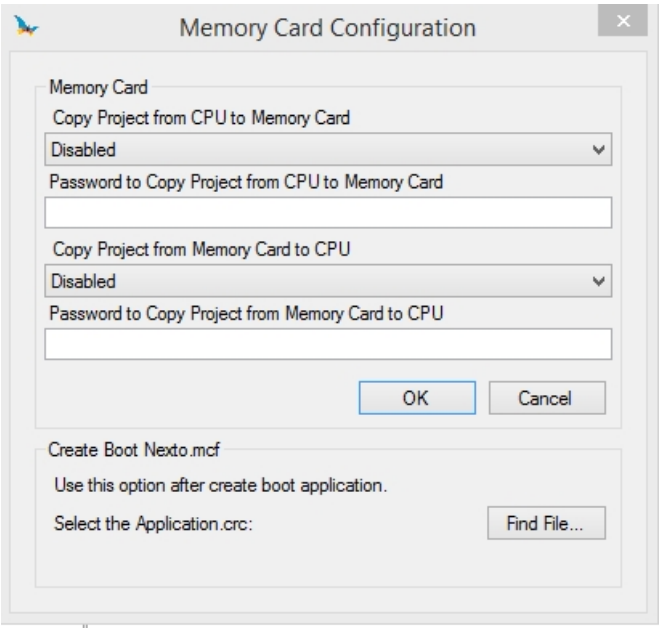


Figure 131: Memory Card Settings

Following these steps, MasterTool IEC XE will send all files needed to perform the send and receive operations of projects via memory card. If the card is mounted, the password will be written to it. Otherwise, the password set in MasterTool will be requested if the user tries to transfer the CPU project to the card.

5.10.2. Project Transfer

To transfer the project from CPU to the memory card or vice versa, the user, in addition to enabling in MasterTool IEC XE software to use the functionality, will have to access the menu *Memory Card* in the CPU, using the diagnostics key, and select the desired transfer option.

ATTENTION

The transfer of the project to the memory card should only be done using the CPU diagnostics key.

Afterwards, you will be prompted for the password if the user has set during application setup. Then with a short press on the diagnostics key the digits are incremented and with a long press are confirmed. In the confirmed sixth digit, the CPU will consist of the password and start the process.

After transferring the memory card to the CPU, if there is a RUN application it will be kept in STOP for safety reasons. To put the CPU in RUN, it must be rebooted.

When the passwords of both the application that is in the CPU and the application that is on the memory card are the same, it is not required to enter the passwords in the CPU menu to perform the application transfers. For more information on using the diagnostics key, see section One Touch Diag.

To remove the memory card, simply press and hold the *MS* key and wait until the memory card icon disappears from the graphic display status screen.

ATTENTION

If the memory card is removed without have been unmounted through CPU's menu, during a file transference, this process can cause the loss of card data as well as corrupt the files in it. This process may cause the need of another card formatting when it'll be inserted on the CPU again.

ATTENTION

If there is any file at memory card root named "*NextoMemCard*" or "*Backup*", it will be deleted to create the system folders with the same name, used by the CPU to store the project application and the project archive. Folders with these names will not be overwritten.

5.10.3. MasterTool Access

The memory card access is connected to the same user memory screen in the MasterTool IEC XE software, being it mounted in the folder called *MemoryCard*. *NextoMemCard* and *Backup* folders are created into the memory card every time the latter is inserted into the CPU. In case these folders already exist, the system will recognize them and will not overwrite the folders.

In the *NextoMemCard* on the memory card, you will find the application files, in this window you still have the option to save your project in a preferred directory (if you have sent the source code). In MasterTool in the option "*File / Project Archive / Extract Archive...*" you can open in MasterTool the saved application, which is located in the directory previously chosen.

The *Backup* folder is not used by the user.

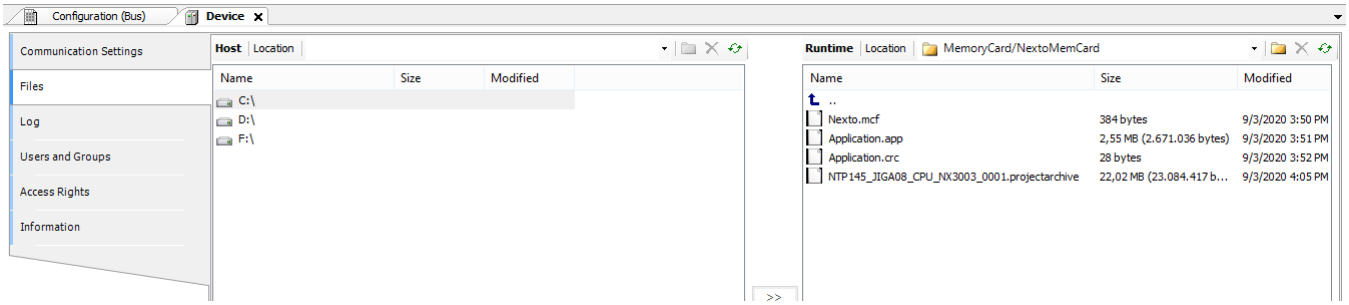


Figure 132: Directory with Memory Card Inserted with Project

ATTENTION

The files transference time depends on the interval time difference minus the average execution time of the task (s) in execution (available time until the next task cycle), it means, the bigger this difference for each task in an application, the faster will be the transference of a data from the memory card to a CPU/MasterTool IEC XE or vice-versa.
Transferring files to the memory card will be slower than the transfer to the internal memory of the CPU. For a CPU in Stop Mode or with no application, the transfer rate is close to 100 Kbytes/s.

5.11. CPU's Informative and Configuration Menu

The access to the *Informative Menu*, the Nexto CPU *Configuration* and the detailed diagnostics, are available through levels and to access the menu information, change level and modify any configuration, a long touch is required on the diagnostic button and to navigate through the items on the same level, a short touch on the diagnostic button is required. See One Touch Diag section to verify the functioning and the difference between the diagnostics button touch types.

The table below shows the menu levels and each screen type available in the CPUs, if they are informative, configurable or to return a level.

Level 1	Level 2	Level 3	Type
HARDWARE	TEMPERATURE	-	Informative
	CONTRAST	CONTRAST LEVEL	Configurable
	DATE AND TIME	-	Informative
	BACK	-	Return level
LANGUAGES	ENGLISH	>ENGLISH	Configurable
	PORTUGUES	>PORTUGUES	Configurable
	ESPANOL	>ESPANOL	Configurable
	BACK	-	Return level
NETWORK	NET 1 IP ADDR.		Informative
	NET 1 MASK		Informative
	NET 2 IP ADDR.	-	Informative
	NET 2 MASK		Informative
	BACK		Return level
REDUNDANCY	PLC ID		Informative
	REMOTE STATE	-	Informative
	PROJ.SYNC.		Informative
	BACK		Return level
SOFTWARE	FIRMWARE		Informative
	BOOTLOADER	-	Informative
	AUX. PROC.		Informative
	BACK		Return level
CARTAO DE MEM.	MEMCARD > CPU	CPU PASSWORD	Configurable
	CPU > MEMCARD	MC PASSWORD	Configurable
	FORMAT	CONFIRM ?	Configurable
	BACK	-	Return level
BACK	-	-	Return level

Table 166: CPU Menu Levels

Notes:

Memory Card: The memory card is only available in the menu, if it is connected in the Nexto CPU.

Redundancy: The "*REDUNDANCY*" menu will only be available in case the NX3030 CPU is identified as Redundant.

Password: The memory card data access password is only necessary in case it is configured in the MasterTool IEC XE software. You cannot edit the password via menu.

As shown on Table 166, between the available options to visualize and modify are the main data necessary to user, as:

- Information about the hardware resources:
  - TEMPERATURE – CPU Internal temperature (Ex.: 36 C 97 F)
  - CONTRAST – Contrast setting of the CPU frontal display
  - DATE AND TIME – Date and time set in the CPU (Ex.: 2001.01.31 00:00)
- Changing the menu language on the CPU:
  - PORTUGUES – Changes the language to Portuguese
  - ENGLISH – Changes the language to English
  - ESPANOL – Changes the language to Spanish
- Visualization of information about the network set in the device:
  - NET 1 IP ADDR. – Address (Ex.: 192.168.0.1)
  - NET 1 MASK – Subnet mask (Ex.: 255.255.255.0)
  - NET 2 IP ADDR – Address (Ex.: 192.168.0.2)

  - NET 2 MASK – Subnet mask (Ex.: 255.255.255.0)
- Access to the PLC redundancy information:
  - PLC ID – Informs the PLC identification in the redundancy. Possible information:
    - PLC A
    - PLC B
  - REMOTE STATE – Informs the state of the remote redundant PLC. Possible states:
    - ACTIVE
    - STANDBY
    - INACTIVE
    - NOT CONFIG.
    - STARTING
    - UNAVAILABLE
  - PROJ. SYNC. – Informs if the synchronization of the projects is enabled
    - CONNECTED
    - NOT CONNEC.
    - DISABLED
    - START SYNC.
    - SYNCHRONIZED
- Information about the software versions:
  - FIRMWARE – CPU software version (Ex.: 1.0.0.0)
  - BOOTLOADER – CPU bootloader version (Ex.: 1.0.0.0)
  - AUX. PROC. – CPU auxiliary processor version (Ex.: 1.0.0.0)
- Access to the Memory Card data:
  - MEMCARD > CPU – Transference of the memory card project to the CPU
  - CPU > MEMCARD – Transference of the CPU project to the memory card
  - FORMAT – Formats the card to the FAT32 files system

The figure below describes an example of how to operate the Nexto CPUs menu through the contrast adjust menu procedure from the *Status*screen. Besides to make the configuration easy, it is possible to identify all screen levels and the touch type to navigate through them, and to modify other parameters as *Language* and the *Memory Card*, using the same access logic. The short touch shows the contrast is being incremented (clearer) and in the next touch after its maximum value, it returns to the minimum value (less clear). The long touch shows the confirmation of the desired contrast and its return to the previous level.

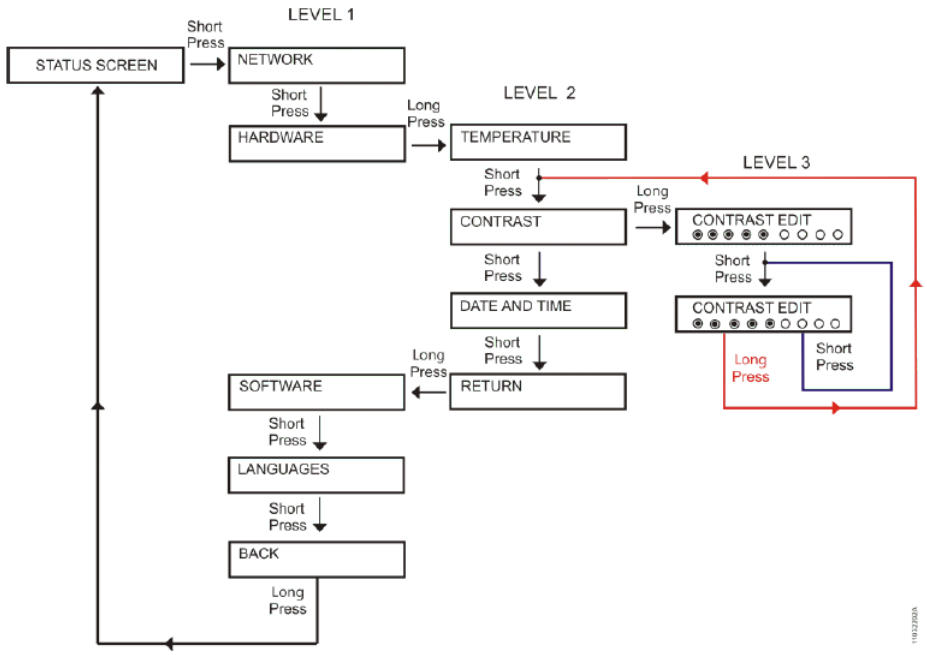


Figure 133: Contrast Adjust

Besides the possibility of the Nexto CPUs menu to be closed through a long touch on the screen diagnostic button *BACK* from level 1, there are also other output conditions that are described below:

- Short touch, at any moment, in the other modules existent on the bus, make the CPU disconnect from the menu and show the desired module diagnostic.
- Idle time, at any level, superior to 5 s.

5.12. Function Blocks and Functions

5.12.1. Special Function Blocks for Serial Interfaces

The special function blocks for serial interfaces make possible the local access (COM 1 AND COM 2) and also access to remote serial ports (expansion modules). Therefore, the user can create his own protocols and handle the serial ports as he wishes, following the IEC 61131-3 languages available in the MasterTool IEC XE software. The blocks are available inside the *NextoSerial* library which must be added to the project so it's possible to use them (to execute the library insertion procedure, see MasterTool IEC XE Programming Manual – MP399609, section Library).

The special function blocks for serial interfaces can take several cycles (consecutive calls) to complete the task execution. Sometimes a block can be completed in a single cycle, but in the general case, needs several cycles. The task execution associated to a block can have many steps which some depend on external events, that can be significantly delayed. The function block cannot implement routines to occupy the time while waits for these events, because it would make the CPU busy. The solution could be the creation of blocking function blocks, but this is not advisable because it would increase the user application complexity, as normally, the multitask programming is not available. Therefore, when an external event is waited, the serial function blocks are finished and the control is returned to the main program. The task treatment continues in the next cycle, in other words, on the next time the block is called.

Before describing the special function blocks for serial interfaces, it is important to know the *Data types*, it means, the data type used by the blocks.

Data type	Options	Description
SERIAL_BAUDRATE	BAUD200	Lists all baud rate possibilities (bits per second)
	BAUD300	
	BAUD600	
	BAUD1200	
	BAUD1800	
	BAUD2400	
	BAUD4800	
	BAUD9600	
	BAUD19200	
	BAUD38400	
	BAUD57600	
	BAUD115200	
SERIAL_DATABITS	DATABITS_5	Lists all data bits possibilities.
	DATABITS_6	
	DATABITS_7	
	DATABITS_8	
SERIAL_HANDSHAKE	Defines all modem signal possibilities for the configurations:	
	RS232_RTS	Controls the Nexto CPU RS-232C port. The transmitter is enabled to start the transmission and disabled as soon as possible after the transmission is finished. For example, can be used to control a RS-232/RS-485 external converter.
	RS232_RTS_OFF	Controls the RS-232C port of the Nexto CPU. The RTS signal is always off.

Data type	Options	Description
	RS232_RTS_ON	Controls the RS-232C port of the Nexto CPU. The RTS signal is always on.
	RS232_RTS_CTS	Controls the RS-232C port of the Nexto CPU. In case the CTS is disabled, the RTS is enabled. Then waits for the CTS to be enabled to get the transmission and RTS restarts as soon as possible, at the end of transmission. Ex: Controlling radio modems with the same modem signal.
	RS232_MANUAL	Controls the RS-232C port of the Nexto CPU. The user is responsible to control all the signals (RTS, DTR, CTS, DSR, DCD).
SERIAL_MODE	NORMAL_MODE	Serial Communication Normal Operation mode.
	EXTENDED_MODE	Serial Communication Extended Operation mode in which are provided information about the received data frame.
SERIAL_PARAMETERS	Defines all configuration parameters of the serial port:	
	BAUDRATE	Defined in SERIAL_BAUDRATE.
	DATABITS	Defined in SERIAL_DATABITS.
	STOPBITS	Defined in SERIAL_STOPBITS.
	PARITY	Defined in SERIAL_PARITY.
	HANDSHAKE	Defined in SERIAL_HANDSHAKE.
	UART_RX_THRESHOLD	Byte quantity which must be received to generate a new UART interruption. Lower values make the TIMESTAMP more precise when the EXTENDED MODE is used and minimizes the overrun errors. However, values too low may cause too many interruptions and delay the CPU.
	MODE	Defined in SERIAL_MODE.
	ENABLE_RX_ON_TX	When true, all the received byte during the transmission will be discharged instead going to the RX line. Used to disable the full-duplex operation in the RS-422 interface.
	ENABLE_DCD_EVENT	When true, generates an external event when the DCD is modified.
	ENABLE_CTS_EVENT	When true, generates an external event when the CTS is modified.
SERIAL_PARITY	PARITY_NONE	List all parity possibilities.
	PARITY_ODD	
	PARITY_EVEN	
	PARITY_MARK	

Data type	Options	Description
	PARITY_SPACE	
SERIAL_PORT	COM 1	List all available serial ports (COM 10, COM 11, COM 12, COM 13, COM 14, COM 15, COM 16, COM 17, COM 18 and COM 19 – expansion modules).
	COM 2	
SERIAL_RX_CHAR_ EXTENDED	Defines a character in the RX queue in extended mode.	
	RX_CHAR	Data byte.
	RX_ERROR	Error code.
	RX_TIMESTAMP	Silence due to the previous character or due to another event which has happen before this character (serial port configuration, transmission ending).
SERIAL_RX_QUEUE_ STATUS	It has some fields which deliver information regarding RX queue status/error, used when the normal format is utilized (no error and timestamp information):	
	RX_FRAMING_ERRORS	Frame errors counter: character incorrect formation – no stop bit, incorrect baud rate, among other – since the serial port configuration. Returns to zero when it reaches the maximum value (65535).
	RX_PARITY_ERRORS	Parity errors counter, since the serial port configuration. Returns to zero when it reaches the maximum value (65535).
	RX_BREAK_ERRORS	Interruption errors counter, since the serial port configuration, in other words, active line higher than the character time. Returns to zero when it reaches the maximum value (65535).
	RX_FIFO_OVERRUN_ ERRORS	FIFO RX overrun errors counter, since the serial port configuration, in other words, error in the FIFO RX configured threshold. Returns to zero when it reaches the maximum value (65535).
	RX_QUEUE_OVERRUN_ ERRORS	RX queue overrun errors counter, in other words, the maximum characters number (1024) was overflowed and the data are being overwritten. Returns to zero when it reaches the maximum value (65535).
	RX_ANY_ERRORS	Sum the last 5 error counters (frame, parity, interruption, RX FIFO overrun, RX queue overrun).
	RX_REMAINING	Number of characters in the RX queue.

Data type	Options	Description
SERIAL_STATUS	List of critic error codes that can be returned by the serial function block. Each block returns specific errors, which will be described below:	
	NO_ERROR	No errors.
	ILLEGAL_*	Return the parameters with invalid values or out of range: - SERIAL_PORT - SERIAL_MODE - BAUDRATE - DATA_BITS - PARITY - STOP_BITS - HANDSHAKE - UART_RX_THRESHOLD - TIMEOUT - TX_BUFF_LENGTH - HANDSHAKE_METHOD - RX_BUFF_LENGTH
	PORT_BUSY	Indicates the serial port is being used by another instance
	HW_ERROR_UART	Hardware error detected in the UART.
	HW_ERROR_REMOTE	Hardware error at communicating with the remote serial port.
	CTS_TIMEOUT_ON	Time-out while waiting for the CTS enabling, in the RS-232 RTS/CTS handshake, in the SERIAL_TX block.
	CTS_TIMEOUT_OFF	Time-out while waiting for the CTS disabling, in the RS-232 RTS/CTS handshake, in the SERIAL_TX block.
	TX_TIMEOUT_ERROR	Time-out while waiting for the transmission ending in the SERIAL_TX.
	RX_TIMEOUT_ERROR	Time-out while waiting for all characters in the SERIAL_RX block or the SERIAL_RX_EXTENDED block.
	FB_SET_CTRL_ NOT_ALLOWED	The SET_CTRL block can't be used in case the handshake is different from RS232_MANUAL.
	FB_GET_CTRL_ NOT_ALLOWED	The GET_CTRL block can't be used in case the handshake is different from RS232_MANUAL.
	FB_SERIAL_RX_ NOT_ALLOWED	The SERIAL_RX isn't available for the RX queue, extended mode.
	FB_SERIAL_RX_ EXTENDED_NOT_ALLOWED	The SERIAL_RX_EXTENDED isn't available for the RX queue, normal mode.
	DCD_INTERRUPT_ NOT_ALLOWED	The interruption by the DCD signal can't be enabled in case the serial port doesn't have the respective pin.

Data type	Options	Description
	CTS_INTERRUPT_ NOT_ALLOWED	The interruption by the CTS signal can't be enabled in case the handshake is different from RS232_MANUAL or in case the serial port doesn't have the respective pin.
	DSR_INTERRUPT_ NOT_ALLOWED	The interruption by the DSR signal can't be enabled in case the serial port doesn't have the respective pin. (Nexto CPUs don't have this signal in local ports)
	NOT_CONFIGURED	The function block can't be used before the serial port configuration.
	INTERNAL_ERROR	Indicates that an internal problem has ocurred in the serial port.
SERIAL_STOPBITS	STOPBITS_1	List all Stop Bits possibilities.
	STOPBITS_2	
	STOPBITS_1_5	

Table 167: Serial Function Blocks Data types

5.12.1.1. SERIAL_CFG

This function block is used to configure and initialize the desired serial port. After the block is called, every RX and TX queue associated to the serial ports and the RX and TX FIFO are restarted.

Figure 134: Serial Configuration Block

Input parameters	Type	Description
REQUEST	BOOL	This variable, when true, enables the function block use.
PORT	SERIAL_PORT	Select the serial port, as described in the SERIAL_PORT data type.
PARAMETERS	SERIAL_PARAMETERS	This structure defines the serial port configuration parameters, as described in the SERIAL_PARAMETERS data type.

Table 168: SERIAL_CFG Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - ILLEGAL_SERIAL_MODE - ILLEGAL_BAUDRATE - ILLEGAL_DATA_BITS - ILLEGAL_PARITY - ILLEGAL_STOP_BITS - ILLEGAL_HANDSHAKE - ILLEGAL_UART_RX_THRESHOLD - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - DCD_INTERRUPT_NOT_ALLOWED - CTS_INTERRUPT_NOT_ALLOWED - DSR_INTERRUPT_NOT_ALLOWED

Table 169: SERIAL_CFG Output Parameters

Utilization example in ST language, after the library Nexto Serial is inserted in the project:

```
PROGRAM UserPrg
VAR
Config: SERIAL_CFG;
Port: SERIAL_PORT := COM1;
Parameters: SERIAL_PARAMETERS := (BAUDRATE := BAUD9600,
DATABITS := DATABITS_8,
STOPBITS := STOPBITS_1,
PARITY := PARITY_NONE,
HANDSHAKE := RS232_RTS,
UART_RX_THRESHOLD := 8,
MODE :=NORMAL_MODE,
ENABLE_RX_ON_TX := FALSE,
ENABLE_DCD_EVENT := FALSE,
ENABLE_CTS_EVENT := FALSE);
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Config.REQUEST := TRUE;
Config.PORT := Port;
```

```
Config.PARAMETERS := Parameters;
//FUNCTION:
Config();
//OUTPUTS:
Config.DONE;
Config.EXEC;
Config.ERROR;
Status := Config.STATUS;    //If it is necessary to treat the error.
```

5.12.1.2. SERIAL_GET_CFG

The function block is used to capture the desired serial port configuration.

Figure 135: Block to Capture the Serial Configuration

Input parameters	Type	Description
REQUEST	BOOL	This variable, when true, enables the function block use.
PORT	SERIAL_PORT	Select the serial port, as described in the SERIAL_PORT data type.

Table 170: SERIAL_GET_CFG Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.

Output parameters	Type	Description
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - NOT_CONFIGURED
PARAMETERS	SERIAL_PARAMETERS	This structure receives the serial port configuration parameters, as described in the SERIAL_PARAMETERS data type.

Table 171: SERIAL_GET_CFG Output Parameters

Utilization example in ST language, after the library is inserted in the project:

```
PROGRAM UserPrg
VAR
GetConfig: SERIAL_GET_CFG;
Port: SERIAL_PORT := COM1;
Parameters: SERIAL_PARAMETERS;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
GetConfig.REQUEST := TRUE;
GetConfig.PORT := Port;
//FUNCTION:
GetConfig();
//OUTPUTS:
GetConfig.DONE;
GetConfig.EXEC;
GetConfig.ERROR;
Status := GetConfig.STATUS; //If it is necessary to treat the error.
Parameters := GetConfig.PARAMETERS; //Receive the parameters of desired serial
   port.
```

5.12.1.3. SERIAL_GET_CTRL

This function block is used to read the CTS, DSR and DCD control signals, in case they are available in the serial port. A false value will be returned when there are not control signals.

Figure 136: Block Used to Visualize the Control Signals

Input parameters	Type	Description
REQUEST	BOOL	This variable, when true, enables the function block use.
PORT	SERIAL_PORT	Select the serial port, as described in the SERIAL_PORT data type.

Table 172: SERIAL_GET_CTRL Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - FB_GET_CTRL_NOT_ALLOWED - NOT_CONFIGURED
CTS_VALUE	BOOL	Value read in the CTS control signal.
DSR_VALUE	BOOL	Value read in the DSR control signal.
DCD_VALUE	BOOL	Value read in the DCD control signal.

Table 173: SERIAL_GET_CTRL Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Get_Control: SERIAL_GET_CTRL;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Get_Control.REQUEST := TRUE;
Get_Control.PORT := Port;
//FUNCTION:
Get_Control();
//OUTPUTS:
Get_Control.DONE;
Get_Control.EXEC;
Get_Control.ERROR;
Status := Get_Control.STATUS;   //If it is necessary to treat the error.
Get_Control.CTS_VALUE;
Get_Control.DSR_VALUE;
Get_Control.DCD_VALUE;
```

5.12.1.4. SERIAL_GET_RX_QUEUE_STATUS

This block is used to read some status information regarding the RX queue, specially developed for the normal mode, but it can also be used in the extended mode.

Figure 137: Block Used to Visualize the RX Queue Status

Input parameters	Type	Description
REQUEST	BOOL	This variable, when true, enables the function block use.
PORT	SERIAL_PORT	Select the serial port, as described in the SERIAL_PORT data type.

Table 174: SERIAL_GET_RX_QUEUE_STATUS Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - NOT_CONFIGURED
RXQ_STATUS	SERIAL_RX_ QUEUE_STATUS	Returns the RX queue status/error, as described in the SERIAL_RX_QUEUE_STATUS data type.

Table 175: SERIAL_GET_RX_QUEUE_STATUS Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Get_Status: SERIAL_GET_RX_QUEUE_STATUS;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
Status_RX: SERIAL_RX_QUEUE_STATUS;
END_VAR
//INPUTS:
Get_Status.REQUEST := TRUE;
Get_Status.PORT := Port;
//FUNCTION:
Get_Status();
//OUTPUTS:
Get_Status.DONE;
Get_Status.EXEC;
Get_Status.ERROR;
Status := Get_Status.STATUS;    //If it is necessary to treat the error.
Status_RX := Get_Status.RXQ_STATUS; //If it is necessary to treat the error of
   the RX.
```

5.12.1.5. SERIAL_PURGE_RX_QUEUE

This function block is used to clean the serial port RX queue, local and remote. The UART RX FIFO is restarted too.

Figure 138: Block Used to Clean the RX Queue

Input parameters	Type	Description
REQUEST	BOOL	This variable, when true, enables the function block use.
PORT	SERIAL_PORT	Select the serial port, as described in the SERIAL_PORT data type.

Table 176: SERIAL_PURGE_RX_QUEUE Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It's false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It's false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It's false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - NOT_CONFIGURED

Table 177: SERIAL_PURGE_RX_QUEUE Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Purge_Queue: SERIAL_PURGE_RX_QUEUE;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Purge_Queue.REQUEST := TRUE;
Purge_Queue.PORT := Port;
//FUNCTION:
Purge_Queue();
//OUTPUTS:
Purge_Queue.DONE;
Purge_Queue.EXEC;
Purge_Queue.ERROR;
Status := Purge_Queue.STATUS; //If it is necessary to treat the error.
```

5.12.1.6. SERIAL_RX

This function block is used to receive a serial port buffer, using the RX queue normal mode. In this mode, each character in the RX queue occupy a single byte which has the received data, storing 5, 6, 7 or 8 bits, according to the serial interface configuration.

Figure 139: Block Used to Read the Reception Buffer Values

Input parameters	Type	Description
REQUEST	BOOL	This variable, when true, enables the function block use.
PORT	SERIAL_PORT	Select the serial port, as described in the SERIAL_PORT data type.
RX_BUFFER_ POINTER	POINTER TO BYTE	Pointer of a byte array to receive the buffer values.
RX_BUFFER_ LENGTH	UINT	Specify the expected character number in the byte array. In case more than the expected bytes are available, only the expected quantity will be read from the byte array, the rest will be leaved in the RX queue (maximum size equal to 1024 characters).

Input parameters	Type	Description
RX_TIMEOUT	UINT	Specify the time-out to receive the expected character quantity. In case it is smaller than the necessary to receive the characters, the RX_TIME-OUT_ERROR output from the STATUS parameter will be indicated. When the specified value, in ms, is equal to zero, the function will return the data within the buffer.

Table 178: SERIAL_RX Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - ILLEGAL_RX_BUFF_LENGTH - RX_TIMEOUT_ERROR - FB_SERIAL_RX_NOT_ALLOWED - NOT_CONFIGURED
RX_RECEIVED	UINT	Returns the received characters number. This number can be within zero and the configured value in RX_BUFFER_LENGTH. In case it is smaller, an error will be indicated by the function block.
RX_REMAINING	UINT	Returns the number of characters which are still in the RX queue after the function block execution.

Table 179: SERIAL_RX Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Receive: SERIAL_RX;
Port: SERIAL_PORT := COM1;
Buffer_Pointer: ARRAY [0..1023] OF BYTE;    //Max size.
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Receive.REQUEST := TRUE;
Receive.PORT := Port;
Receive.RX_BUFFER_POINTER := ADR(Buffer_Pointer);
Receive.RX_BUFFER_LENGTH := 1024;     //Max size.
Receive.RX_TIMEOUT := 10000;
//FUNCTION:
Receive();
//OUTPUTS:
Receive.DONE;
Receive.EXEC;
Receive.ERROR;
Status := Receive.STATUS;   //If it is necessary to treat the error.
Receive.RX_RECEIVED;
Receive.RX_REMAINING;
```

5.12.1.7. SERIAL_RX_EXTENDED

This function block is used to receive a serial port buffer using the RX queue extended mode as shown in the Serial Interfaces Configuration section.

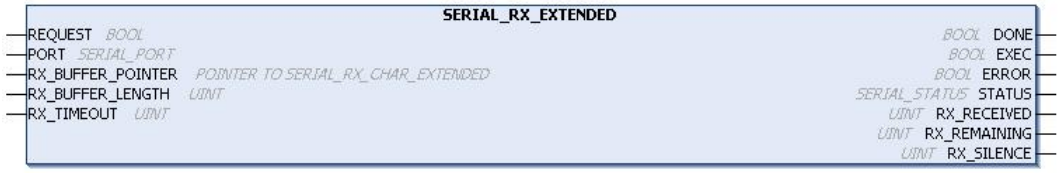


Figure 140: Block Used for Reception Buffer Reading

Input parameters	Type	Description
REQUEST	BOOL	This variable, when true, enables the function block use.
PORT	SERIAL_PORT	Select the serial port, as described in the SERIAL_PORT data type.
RX_BUFFER_ POINTER	POINTER TO SERIAL_RX_CHAR _EXTENDED	Pointer of a SERIAL_RX_CHAR_EXTENDED array to receive the buffer values.

Input parameters	Type	Description
RX_BUFFER_ LENGTH	UINT	Specify the expected character number in the SERIAL_RX_CHAR_EXTENDED array. In case more than the expected bytes are available, only the expected quantity will be read from the byte array, the rest will be leaved in the RX queue (maximum size equal to 1024 characters).
RX_TIMEOUT	UINT	Specify the time-out to receive the expected character quantity. In case it is smaller than the necessary to receive the characters, the RX_TIMEOUT_ERROR output from the STATUS parameter will be indicated. When the specified value, in ms, is equal to zero, the function will return the data within the buffer.

Table 180: SERIAL_RX_EXTENDED Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - ILLEGAL_RX_BUFF_LENGTH - RX_TIMEOUT_ERROR - FB_SERIAL_RX_EXTENDED_NOT _ALLOWED - NOT_CONFIGURED
RX_RECEIVED	UINT	Returns the received characters number. This number can be within zero and the configured value in RX_BUFFER_LENGTH. In case it is smaller, an error will be indicated by the function block.

Output parameters	Type	Description
RX_REMAINING	UINT	Returns the number of characters which are still in the RX queue after the function block execution.
RX_SILENCE	UINT	Returns the silence time in the RX queue, measured since the last received character is finished. The time unit is 10 μs. This output parameter type is important to detect the silence time in protocols as MODBUS RTU. It might not be the silence time after the last received character by this function block, as it is only true if RX_REMAINING = 0.

Table 181: SERIAL_RX_EXTENDED Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Receive_Ex: SERIAL_RX_EXTENDED;
Port: SERIAL_PORT := COM1;
Buffer_Pointer: ARRAY [0..1023] OF SERIAL_RX_CHAR_EXTENDED;
Status: SERIAL_STATUS;
END_VAR
//INPUTS:
Receive_Ex.REQUEST := TRUE;
Receive_Ex.PORT := Port;
Receive_Ex.RX_BUFFER_POINTER := ADR(Buffer_Pointer);
Receive_Ex.RX_BUFFER_LENGTH := 1024;      //Max size.
Receive_Ex.RX_TIMEOUT := 10000;
//FUNCTION:
Receive_Ex();
//OUTPUTS:
Receive_Ex.DONE;
Receive_Ex.EXEC;
Receive_Ex.ERROR;
Status := Receive_Ex.STATUS;  //If it is necessary to treat the error.
Receive_Ex.RX_RECEIVED;
Receive_Ex.RX_REMAINING;
Receive_Ex.RX_SILENCE;
```

5.12.1.8. SERIAL_SET_CTRL

This block is used to write on the control signals (RTS and DTR), when they are available in the serial port. It can also set a busy condition for the transmission, through BREAK parameter and it can only be used if the modem signal is configured for RS232_MANUAL.

Figure 141: Block for Control Signals Writing

Input parameters	Type	Description
REQUEST	BOOL	This variable, when true, enables the function block use.
PORT	SERIAL_PORT	Select the serial port, as described in the SERIAL_PORT data type.
RTS_VALUE	BOOL	Value to be written on RTS signal.
RTS_EN	BOOL	Enables the RTS_VALUE parameter writing.
DTR_VALUE	BOOL	Value to be written on DTR signal.
DTR_EN	BOOL	Enables the DTR_VALUE parameter writing.
BREAK	BOOL	In case it's true, enables logic 0 (busy) in the transmission line.

Table 182: SERIAL_SET_CTRL Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - FB_SET_CTRL_NOT_ALLOWED - NOT_CONFIGURED

Table 183: SERIAL_SET_CTRL Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Set_Control: SERIAL_SET_CTRL;
Port: SERIAL_PORT := COM1;
Status: SERIAL_STATUS;
END_VAR

//INPUTS:
Set_Control.REQUEST := TRUE;
Set_Control.PORT := Port;
Set_Control.RTS_VALUE := FALSE;
Set_Control.RTS_EN := FALSE;
Set_Control.DTR_VALUE := FALSE;
Set_Control.DTR_EN := FALSE;
Set_Control.BREAK := FALSE;
//FUNCTION:
Set_Control();
//OUTPUTS:
Set_Control.DONE;
Set_Control.EXEC;
Set_Control.ERROR;
Status := Set_Control.STATUS;    //If it is necessary to treat the error.
```

5.12.1.9. SERIAL_TX

This function block is used to transmit a data buffer through serial port and it is only finalized after all bytes were transmitted or after time-out (generating errors).

Figure 142: Block for Values Transmission by the Serial

Input parameters	Type	Description
REQUEST	BOOL	This variable, when true, enables the function block use.
PORT	SERIAL_PORT	Select the serial port, as described in the SERIAL_PORT data type.
TX_BUFFER_ POINTER	POINTER TO BYTE	Pointer of a byte array to transmit the buffer values.
TX_BUFFER_ LENGTH	UINT	Specify the expected character number in the byte array to be transmitted (TX queue maximum size is 1024 characters).

Input parameters	Type	Description
TX_TIMEOUT	UINT	Specify the time-out to complete the transmission including the handshake phase. The specified value, in ms, must be positive and different than zero.
DELAY_BEFORE_TX	UINT	Specify the delay, in ms, between the function block call and the transmission beginning. This variable can be used in communications with some modems.
CLEAR_RX_BEFORE_TX	BOOL	When true, the RX queue and the UART FIFO RX are erased before the transmission beginning. This behavior is typical in half-duplex master/slave protocols.

Table 184: SERIAL_TX Input Parameters

Output parameters	Type	Description
DONE	BOOL	This variable is true when the block is completely executed. It is false otherwise.
EXEC	BOOL	This variable is true while the block is being executed. It is false otherwise.
ERROR	BOOL	This variable is true when the block concludes the execution with an error. It is false otherwise. It is connected to the variable DONE, as its status is showed after the block conclusion.
STATUS	SERIAL_STATUS	In case the ERROR variable is true, the STATUS structure will show the error found during the block execution. The possible states, already described in the SERIAL_STATUS data type, are: - NO_ERROR - ILLEGAL_SERIAL_PORT - PORT_BUSY - HW_ERROR_UART - HW_ERROR_REMOTE - ILLEGAL_TX_BUFF_LENGTH - ILLEGAL_TIMEOUT - CTS_TIMEOUT_ON - CTS_TIMEOUT_OFF - TX_TIMEOUT_ERROR - NOT_CONFIGURED
TX_TRANSMITTED	UINT	Returns the transmitted byte number which must be equal to TX_BUFFER_LENGTH, but can be smaller in case some error has occurred during transmission.

Table 185: SERIAL_TX Output Parameters

Utilization example in ST language, after the library is inserted in the project and the serial port configured:

```
PROGRAM UserPrg
VAR
Transmit: SERIAL_TX;
Port: SERIAL_PORT := COM1;
Buffer_Pointer: ARRAY [0..9] OF BYTE := [0,1,2,3,4,5,6,7,8,9];
Status: SERIAL_STATUS;
END_VAR

//INPUTS:
Transmit.REQUEST := TRUE;
Transmit.PORT := Port;
Transmit.TX_BUFFER_POINTER := ADR(Buffer_Pointer);
Transmit.TX_BUFFER_LENGTH := 10;
Transmit.TX_TIMEOUT := 10000;
Transmit.DELAY_BEFORE_TX := 1000;
Transmit.CLEAR_RX_BEFORE_TX := TRUE;
//FUNCTION:
Transmit();
//OUTPUTS:
Transmit.DONE;
Transmit.EXEC;
Transmit.ERROR;
Status := Transmit.STATUS;    //If it is necessary to treat the error.
Transmit.TX_TRANSMITTED;
```

5.12.2. Inputs and Outputs Update

By default, the local bus and CPU integrated I/O are updated on every execution cycle of MainTask. The Refresh functions allows to update these I/O points asynchronously at any point of user application code.

When the function blocks to update the inputs and outputs are not used, the update is performed every cycle of the MainTask.

ATTENTION

At the startup of a CPU of this series, the inputs and outputs are only updated for reading and prepared for writing when the MainTask is performed.
All other system tasks that run before MainTask will be with the inputs and outputs invalid.

5.12.2.1. REFRESH_INPUT

This function block is used to update the specified module inputs without the necessity to wait for the cycle to be completed. It is important to notice that the filters configured in the MasterTool IEC XE and the update time of the module inputs will have to be considered in effective time of the inputs update in the application developed by the user.

ATTENTION

The *REFRESH_INPUT* function must only be used in MainTask.
To update inputs in other tasks, the option *Enable I/O update per task* must be selected, for further information about this option, consult Table 44.

ATTENTION

REFRESH_INPUT function does not support inputs that have been mapped to symbolic variables. For proper operation it is necessary that the input is mapped to a variable within the memory direct representation of input variables (%I).

ATTENTION

The *REFRESH_INPUT* function updates only the direct variables %I that are declared in the "Bus: I/O Mapping" tab of the module addressed in the respective rack/slot of the function. In the case of communication modules/interfaces (MODBUS, Profibus, etc.), the update does not include the direct variables of the device mappings.

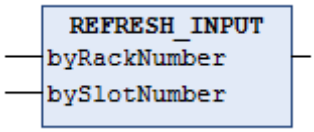


Figure 143: Block for Input Updating

Input parameters	Type	Description
byRackNumber	BYTE	Rack number.
bySlotNumber	BYTE	Position number where the module is connected.

Table 186: REFRESH_INPUT Input Parameters

Possible *TYPE_RESULT*:

- **OK_SUCCESS:** Execution success.
- **ERROR_FAILED:** This error is returned if the function is called for a module that has only outputs, or also if the option *Always update variables* (located in the module's configuration screen, tab *I/O Mapping*) is not checked.
- **ERROR_NOTSUPPORTED:** The called routine is not supported by the product.
- **ERROR_PARAMETER:** Invalid / unsupported parameter.
- **ERROR_MODULE_ABSENT:** The module is absent in the bus.
- **ERROR_MODULE_NOTCONFIGURED:** The module is not configured in the application.
- **ERROR_MODULE_NOTRUNNING:** The module is not running (is not in operational state).
- **ERROR_MODULE_COMMFAIL:** Failure in the communication with the module.
- **ERROR_MODULE_NOTFOUND:** The module was not found in the application or is not supported.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
Info: TYPE_RESULT;
byRackNumber: BYTE;
bySlotNumber: BYTE;
END_VAR
//INPUTS:
byRackNumber := 0;
bySlotNumber := 10;
//FUNCTION:
Info := REFRESH_INPUT (byRackNumber, bySlotNumber); //Function call.
//Variable "Info" receives possible function errors.
```

5.12.2.2. REFRESH_OUTPUT

This function block is used to update the specified module outputs. It is not necessary to wait until the cycle is finished. It is important to notice that the update time of the module outputs will have to be considered in the effective time of the outputs update in the application developed by the user.

ATTENTION

The *REFRESH_OUTPUT* function must only be used in MainTask.
To update outputs in other tasks, the option *Enable I/O update per task* must be selected, for further information about this option, consult Table 44.

ATTENTION

REFRESH_OUTPUT function does not support inputs that have been mapped to symbolic variables. For proper operation it is necessary that the input is mapped to a variable within the memory direct representation of input variables (%Q).

ATTENTION

The *REFRESH_OUTPUT* function updates only the direct variables %Q that are declared in the "Bus: I/O Mapping" tab of the module addressed in the respective rack/slot of the function. In the case of communication modules/interfaces (MODBUS, Profibus, etc.), the update does not include the direct variables of the device mappings.

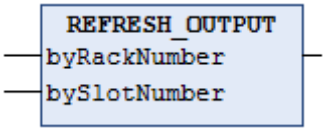


Figure 144: Block for Output Updating

Input parameters	Type	Description
byRackNumber	BYTE	Rack number.
bySlotNumber	BYTE	Position number where the module is connected.

Table 187: REFRESH_OUTPUT Input Parameters

Possible *TYPE_RESULT*:

- **OK_SUCCESS:** Execution success.
- **ERROR_FAILED:** This error is returned if the function is called for a module that has only inputs, or also if the option *Always update variables* (located in the module's configuration screen, tab *I/O Mapping*) is not checked.
- **ERROR_NOTSUPPORTED:** The called routine is not supported by the product.
- **ERROR_PARAMETER:** Invalid / unsupported parameter.
- **ERROR_MODULE_ABSENT:** The module is absent in the bus.
- **ERROR_MODULE_NOTCONFIGURED:** The module is not configured in the application.
- **ERROR_MODULE_NOTRUNNING:** The module is not running (is not in operational state).
- **ERROR_MODULE_COMMFAIL:** Failure in the communication with the module.
- **ERROR_MODULE_NOTFOUND:** The module was not found in the application or is not supported.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
Info: TYPE_RESULT;
byRackNumber: BYTE;
bySlotNumber: BYTE;
END_VAR
//INPUTS:
byRackNumber := 0;
bySlotNumber := 10;
//FUNCTION:
//Function call.
Info := REFRESH_OUTPUT (byRackNumber, bySlotNumber);
//Variable "Info" receives possible function errors.
```

5.12.3. PID Function Block

ATTENTION

The PID function block described up to previous revision L of this manual became obsolete and was removed from this manual.
The PID, PID_INT and PID_REAL function blocks described up to revision C of MP399609, also became obsolete and were also removed from newer versions of that manual. Users that need description of these obsolete function blocks due to maintenance reasons must use revision C of MP399609.
Function blocks PID, PID_INT and PID_REAL must not be used in new projects. These function blocks were replaced by newer function blocks with additional resources, like auto-tuning and support to cascade, override and feed-forward controls. These new function blocks are described in MU214610, and are available after version 1.1.0.0 of library *NextoPID*.

5.12.4. Timer Retain

The timer retain is a function block developed for applications as production line clocks, that need to store its value and restart the counting from the same point in case of power supply failure. The values stored by the function block, are only zero in case of a *Reset Cold*, *Reset Origin* or a new application *Download* (see the MasterTool IEC XE User Manual - MU299609), when the counters keep working, even when the application is stopped (Stop Mode).

ATTENTION

It is important to stress that, for the correct functioning of the Timer Retain blocks, the variables must be declared as Retain (*VAR RETAIN*). It's also important to notice that in simulation mode, the Timer Retain function blocks do not run properly due to need the Nexto CPU for correct behavior.

The three blocks already available in the MasterTool IEC XE software *NextoStandard* library are described below (for the library insertion proceeding, see MasterTool IEC XE Programming Manual – MP399609, section Library).

5.12.4.1. TOF_RET

The function block *TOF_RET* implements a time delay to disable an output. When the input *IN* has its state changed from (TRUE) to (FALSE), or a falling edge, the specified time *PT* will be counted and the *Q* output will be driven to (FALSE) at the end of it. When the input *IN* is in logic level 1 (TRUE), the output *Q* remain in the same state (TRUE), even if this happened in the middle of the counting process. The *PT* time can be changed during the counting as the block assumes the new value if the counting hasn't finished. Figure 145 depicts the *TOF_RET* block and Figure 146 shows its graphic behavior.

Figure 145: TOF_RET Block

Input parameters	Type	Description
IN	BOOL	This variable, when receives a falling edge, enables the block counting.
PT	TIME	This variable specifies the block counting limit (time delay).

Table 188: TOF_RET Input Parameters

Output parameters	Type	Description
Q	BOOL	This variable executes a falling edge as the PT variable (time delay) reaches its maximum value.
ET	TIME	This variable shows the current time delay.

Table 189: TOF_RET Output Parameters

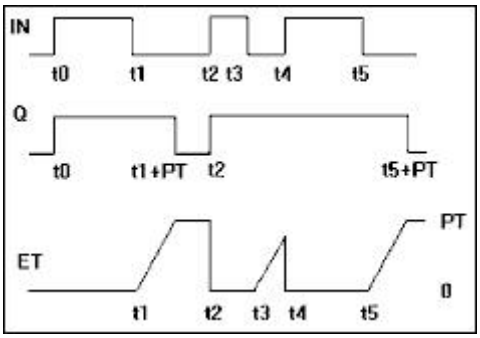


Figure 146: TOF_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM UserPrg
VAR RETAIN
bStart : BOOL := TRUE;
TOF_RET : TOF_RET;
END_VAR

// When bStart=FALSE starts counting
TOF_RET( IN := bStart,
PT := T#20S);

// Actions executed at the end of the counting
IF (TOF_RET.Q = FALSE) THEN
bStart := TRUE;
END_IF
```

5.12.4.2. TON_RET

The *TON_RET* implements a time delay to enable an output. When the input *IN* has its state changed from (FALSE) to (TRUE), or a rising edge, the specified time *PT* will be counted and the *Q* output will be driven to (TRUE) at the end of it. When the input *IN* is in logic level 0 (FALSE), the output *Q* remain in the same state (FALSE), even if it happens in the middle of the counting process. The *PT* time can be changed during the counting as the block assumes the new value if the counting hasn't finished. Figure 147 depicts the *TON_RET* block and Figure 148 shows its graphic behavior.

Figure 147: TON_RET Function Block

Input parameters	Type	Description
IN	BOOL	This variable, when receives a rising edge, enables the function block counting.
PT	TIME	This variable specifies the block counting limit (time delay).

Table 190: TON_RET Input Parameters

Output parameters	Type	Description
Q	BOOL	This variable executes a rising edge as the PT variable (time delay) reaches its maximum value.
ET	TIME	This variable shows the current time delay.

Table 191: TON_RET Output Parameters

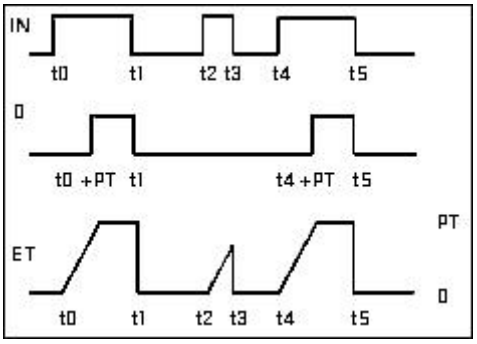


Figure 148: TON_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM UserPrg
VAR RETAIN
bStart : BOOL;
TON_RET : TON_RET;
END_VAR

// Quando bStart=TRUE starts counting
TON_RET( IN := bStart,
PT := T#20S);

// Actions executed at the end of the counting
IF (TON_RET.Q = TRUE) THEN
bStart := FALSE;
END_IF
```

5.12.4.3. TP_RET

The *TP_RET* function block works as a trigger. The timer which starts when the *IN* input has its state changed from (FALSE) to (TRUE), that is, a rising edge, it is increased until the *PT* time limit is reached. During the counting, the *Q* output is (TRUE), otherwise it is (FALSE). The *PT* time can be changed during the counting as the block assumes the new value if the counting has not finished. Figure 149 depicts the *TP_RET* and Figure 150 shows its graphic behavior.

Figure 149: TP_RET Function Block

Input parameters	Type	Description
IN	BOOL	This variable, when receives a rising edge, enables the function block counting.
PT	TIME	This variable specifies the function block counting limit (time delay).

Table 192: TP_RET Input Parameters

Output parameters	Type	Description
Q	BOOL	This variable is true during the counting, otherwise is false.
ET	TIME	This variable shows the current time delay.

Table 193: TP_RET Output Parameters

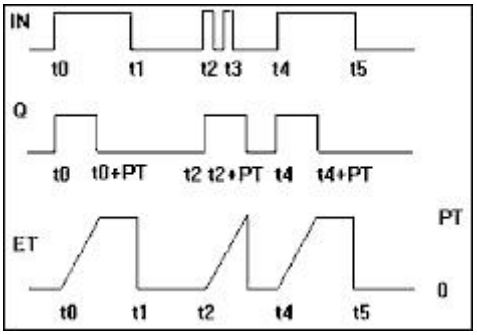


Figure 150: TP_RET Block Graphic Behavior

Utilization example in ST language:

```
PROGRAM UserPrg
VAR RETAIN
bStart : BOOL;
TP_RET : TP_RET;
END_VAR

// Configure TP_RET
TP_RET( IN := bStart,
PT := T#20S);

bStart := FALSE;

// Actions executed during the counting
IF (TP_RET.Q = TRUE) THEN
// Executes while the counter is activated
ELSE
// Executes when the counter is deactivated
END_IF
```

5.12.5. Non-Redundant Timer

The non-redundant timer is used in applications for the redundant NX3030 CPU which need a timer in the non-redundant program of a half-cluster. This timer does not use the IEC timer, therefore, it will not be synchronized in case the reserve half-cluster assumes the active status and the active one goes for reserve.

The three types of blocks already available in the *NextoStandard* library of the MasterTool IEC XE software are describe as follows (for doing the procedure of library's inclusion, check MasterTool IEC XE Programming Manual – MP399608, section Library).

5.12.5.1. TOF_NR

The *TOF_NR* function block implements a delay time for disabling an output and has its functioning and configuration similar to the TOF_RET function block, differentiating itself only for not being redundant nor retentive.

Figure 151: TOF_NR Function Block

Utilization example in ST language:

```
PROGRAM NonSkippedPrg
VAR
bStart : BOOL := TRUE;
TOF_NR : TOF_NR;
END_VAR

// When bStart=FALSE starts the counting
TOF_NR( IN := bStart,
PT := T#20S);

// Actions executed at the end of the counting
IF (TOF_NR.Q = FALSE) THEN
bStart := TRUE;
END_IF
```

5.12.5.2. TON_NR

The *TON_NR* function block implements a delay time to enable an output and has its functioning and configuration similar to the TON_RET function block, differentiating only for not being redundant nor retentive.

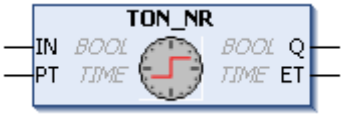


Figure 152: TON_NR Function Block

Utilization example in ST language:

```
PROGRAM NonSkippedPrg
VAR
bStart : BOOL;
TON_NR : TON_NR;
END_VAR

// When bStart=TRUE starts the counting
TON_NR( IN := bStart,
PT := T#20S);

// Actions executed at the end of the counting
IF (TON_NR.Q = TRUE) THEN
bStart := FALSE;
END_IF
```

5.12.5.3. TP_NR

The *TP_NR* function block works as a trigger and has its functioning and configuration similar to the TP_RET function block, differentiating only for not being redundant nor retentive.

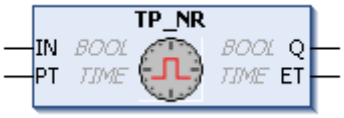


Figure 153: TP_NR Function Block

Utilization example in ST language:

```
PROGRAM NonSkippedPrg
VAR
bStart : BOOL;
TP_NR : TP_NR;
END_VAR

// Configure TP_NR
TP_NR( IN := bStart,
PT := T#20S);

bStart := FALSE;

// Actions executed during the counting
IF (TP_NR.Q = TRUE) THEN
// Executes while the counter is activated
ELSE
// Executes when the counter is deactivated
END_IF
```

5.12.6. User Log

Feature that allows the user to create own records and write to log files on the memory card present in the CPU. The files are generated in a specific directory of the memory card in the CSV format, allowing viewing in text editors and spreadsheets. The separator was the semicolon character. For more information about the use of the memory card, see section Memory Card.

There are two functions available, one for log information and another to remove all records. The following is a description of the types of data used by the functions:

Data type	Option	Description
USER_LOG_EVENT_TYPES	USER_LOG_EVENT_ERROR	The user is free to use the best indication according to log message severity.
	USER_LOG_EVENT_DEBUG	
	USER_LOG_EVENT_INFO	
	USER_LOG_EVENT_WARN	
USER_LOG_MESSAGE		Log message with 150-character max.
USER_LOG_ERROR_CODES	USER_LOG_OK	The operation was performed successfully.
	USER_LOG_FAILED	The operation was not performed successfully. The reason for the failure can be checked in the system logs – see section System Log.
	USER_LOG_BUFFER_FULL	Messages are being added beyond the processing capacity.
	USER_LOG_NO_MEMORY	At the time, there were no resources to perform the operation.
	USER_LOG_FILE_SYSTEM_ERROR	There was an error while accessing the memory card or there is no available space. Error information can be verified in the logs of system – see section System Log.
	USER_LOG_NO_MEMORY_CARD	There is a memory card present in the CPU.
	USER_LOG_MEMORY_CARD_FULL	There is no free space available on the memory card.
	USER_LOG_PROCESSING	The resource is busy executing the last operation, for example, deleting all log files.

Table 194: Data Type for User Log

The following are described the two functions available in the *LibLogs* library on MasterTool IEC XE. To perform the procedure of inserting a library, see the MasterTool IEC Programming Manual – MP399609, section Libraries.

ATTENTION

The User Logs are available only until version 1.3.0.20 of Nexto Series CPUs. In the same way to use this feature is necessary version 1.40 or higher of MasterTool IEC XE.

5.12.6.1. UserLogAdd

This function is used to add a new user log message, adding in a new line to the log file on the memory card. The message must have a maximum length of 150 characters, and the event type of the message. Application variables can be registered using conversion to string and concatenation with the main message. The date and time information in UTC (timestamp) is automatically added in the message with a resolution of milliseconds where the event was registered. The date and time information is also used in the formation of the names of the log files.

The *UserLogAdd* function can be used to enter multiple messages within a single task and also in different application tasks. However independent of each execution of the function in the application, being on the same task or on different tasks, all use the same feature to record the desired messages. For this reason it is recommended that the addition of messages using the *UserLogAdd* function in the application be held every 50 ms to prevent the return of buffer overload. If the function is performed in periods shorter than the indicated, but respect the average time of 50 ms between each message addition at the end of the interval for the task, also prevents the return of buffer overload. So that the logs are added correctly, it is important to respect time limits when the card is inserted and at startup of the CPU, mentioned in section Memory Card. After the operation the function returns the options for the given type *USER_LOG_ERROR_CODES* as Table 194. When the function return is not *USER_LOG_OK*, the message was not registered and the function *UserLogAdd* should be re-executed with the desired message. In case of return of consecutive writing failures, the memory card can be damaged. The replacement by a healthy memory card ensures that the latest logged messages will be recorded on the card that is not damaged, since the CPU is not restarted.

The figure below represents the function *UserLogAdd* and table below the input parameters:

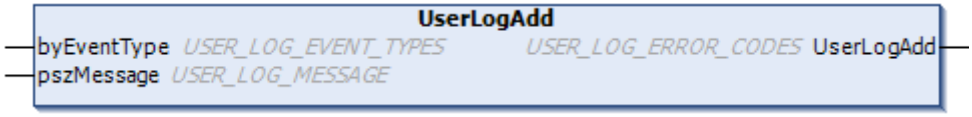


Figure 154: UserLogAdd Function

Input Parameters	Type	Description
byEventType	BYTE	This variable specifies the event type of the log being added as options for the USER_LOG_EVENT_TYPES data type.
pszMessage	USER_LOG_MESSAGE	This variable should contain the set of characters that compose the message to be added to the log file. The message must contain a maximum of 150 characters.

Table 195: UserLogAdd Input Parameters

The log files are generated and organized on the memory card in a specific directory path depending on the CPU serial number and the firmware version installed. For example, for a CPU with serial number 445627 and firmware version 1.4.0.4, the location where the log files should be written to the memory card is *MemoryCard/UserLog/445627/1.4.0.4/*.

The names of the log files are formed by the date and time (timestamp) of the first message. Except when there's a problem to use this name, for example, another existing file with the same name, in this situation it is used the instant date and time. The file name follows the following pattern: *year/month/day/hour/minute/second/millisecond.CSV*. In case of file access problem due to defective sector not enabling to continue writing, will be added to the name of this file the extension *"corrupted"* and a new file will be created. The amount of logs per file is not fixed, varying depending on the size of messages. The amount of created files is limited to 1024 with maximum size of 1 MB each, so the memory card requires 1 GB of free space. When it reaches the limit of 1024 files created on the memory card, during CPU operation, the oldest files are removed so that files with latest logs are preserved, even in cases of partial manual removal of the files in the directory where the files are being written.

The viewing of the log files can be performed through worksheets or conventional text editors. The concatenated information, for improved visualization, may use semicolons between the strings of the message to separate them. One must be careful in formatting cells with floating point values.

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
  eLogError : USER_LOG_ERROR_CODES;
  sMessage :USER_LOG_MESSAGE;
END_VAR

IF (m_rTemperature > MAX_TEMPERATURE_ACCEPT) THEN
  sMessage := 'Temperature higher than expected: ';
  sMessage := concat(sMessage, REAL_TO_STRING(m_rTemperature));
```

```
    sMessage := concat(sMessage, '◦');
    eLogError := UserLogAdd(USER_LOG_EVENT_WARN, sMessage);
  //eLogError variable gets possible function errors.
  END_IF
```

Log file content example: (*UserLog-201308271506245738.csv*)

```
Model; NX3008
Serial number; 445627
Firmware version; 1.4.0.4

27/08/2013 15:06:24.5738; WARN; Overtemperature: 25◦
27/08/2013 16:37:45.3476; WARN; Overtemperature: 25◦
28/08/2013 09:10:55.4201; WARN; Overtemperature: 26◦
```

5.12.6.2. UserLogDeleteAll

The *UserLogDeleteAll* function performs the deletion of log files present in the directory created specifically for the CPU in which is inserted the memory card, i.e. are only deleted the logs contained in the directory named with the CPU firmware version that exists within the directory with the CPU serial version. The log files deleted are only files that exist at the time of memory card mounting and the generated by the *UserLogAdd* function. Logs of other CPUs and files added manually by the user during execution are not deleted. The figure below represents the function *UserLogDeleteAll*.

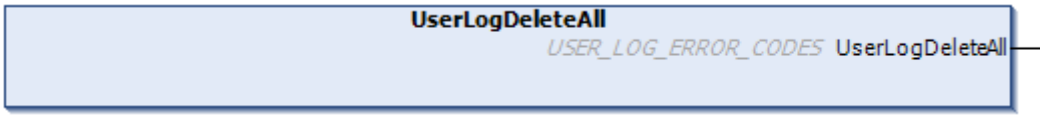


Figure 155: UserLogDeleteAll Function

Utilization example in ST language:

```
PROGRAM UserPrg
VAR
eLogError : USER_LOG_ERROR_CODES;
END_VAR

IF (m_DeleteLogs = TRUE) THEN
eLogError := UserLogDeleteAll();
m_DeleteLogs := FALSE;
//eLogError variable gets possibles function errors.
END_IF
```

ATTENTION

The *UserLogDeleteAll* function's return does not indicate operation completed, just confirmation of execution that can take a large amount of time if there are hundreds of log files in the directory. The function to record the new user log is unavailable right now, returning the *USER_LOG_PROCESSING* option for any operation. The result of the operation can also be checked in the system log.

5.12.7. ClearRtuDiagnostic

This function isn't supported by this CPUs' Series.

5.12.8. ClearEventQueue

The *ClearEventQueue* function available by the *LibRtuStandard* library can be used when it's needed to clear the CPU's event queue and of all instanced drivers.

According to table below the function's execution result is going to be showed in its output variable.

ATTENTION

The *ClearEventQueue* function does not apply to clearing the Event Log (SOE) service queue, described in section SOE Configuration. The function only clears the event queues of the drivers configured under the communication interfaces (COMs and NETs) of the CPU.

Name	ENUM (UDINT)	Result Description
OK_SUCCESS	0	Success
ERROR_FAILED	1	General error
ERROR_NOTSUPPORTED	2	The called routine is not supported by the product
ERROR_PARAMETER	3	Invalid/unsupported parameter
ERROR_MODULE_ABSENT	16	The module is absent in the bus
ERROR_MODULE_NOTCONFIGURED	17	The module is not configured in the application
ERROR_MODULE_NOTRUNNING	18	The module is not running (isn't in operational state)
ERROR_MODULE_COMMFAIL	19	Failure in the communication with the module
ERROR_MODULE_NOTFOUND	20	The module wasn't found in application or is not supported

Table 196: ClearEventQueue Function Results

Using example in ST language, where the function call is going to clear the events queue, and consequently, reset the communication drivers events queue usage diagnostics *T_DIAG_DNP_SERVER_1.tClient_*.tQueueDiags.wUsage:*

```
PROGRAM UserPrg
VAR
  ClearEventQueueStatus : TYPE_RESULT;
END_VAR

ClearEventQueueStatus := ClearEventQueue();
```

5.13. SNMP

5.13.1. Introduction

SNMP (*Simple Network Management Protocol*) is a protocol widely used by network administrators to provide important information and diagnostic equipment present in a given Ethernet network.

This protocol uses the concept of agent and manager, in which the manager sends read requests or write certain objects to the agent. Through a MIB (*Management Information Base*) the manager is aware of existing objects in the agent, and thus can make requests of these objects, respecting the read permissions or writing the same. MIB is a collection of information organized hierarchically with each object of this tree is called OID (*Object Identifier*).

For all equipment with SNMP, it is mandatory to support MIB-II. In this MIB are described key information for managing Ethernet networks.

5.13.2. SNMP nas UCPs Nexto

The CPUs of the Nexto Series behave as agents in SNMP communication. The information made available through SNMP cannot be manipulated or accessed through the user application, requiring an external SNMP manager to perform access. The table below contains the objects available in the Nexto CPUs. This feature is available after firmware version 1.4.0.33 of the Nexto Series CPUs supports the protocols SNMPv1, SNMPv2c and SNMPv3, and support the MIB-II, where objects are described in RFC-1213.

OID	Name	Description
1.3.6.1.2.1.1	System	Contains name, description, location and other equipment identification information.
1.3.6.1.2.1.2	Interfaces	Contains information of the machine's network interfaces. The ifTable (OID 1.3.6.1.2.1.2.2) has the indexes 6 and 7 available, which can be viewed by the network interfaces statistics NET 1 and NET 2, respectively, of the Nexto Series CPUs.
1.3.6.1.2.1.3	At	Contains information of the last required connections to the agent.
1.3.6.1.2.1.4	IP	Contains statistical connections using IP protocol.
1.3.6.1.2.1.5	ICMP	Contains statistical connections using ICMP protocol.
1.3.6.1.2.1.6	TCP	Contains statistical connections using TCP protocol.
1.3.6.1.2.1.7	UDP	Contains statistical connections using UDP protocol.
1.3.6.1.2.1.11	SNMP	Contains statistical connections using SNMP protocol.

Table 197: MIB II Objects – Nexto Series SNMP Agent

By default, the SNMP agent is activated, i.e., the service is initialized at the time the CPU is started. The access to the agent information is via the Ethernet interfaces of the Nexto Series CPUs on UDP port 161. So when the service is active, the agent information can be accessed through any one of the Ethernet interfaces available. It is not possible to provide agent information through Ethernet interfaces NX5000 modules. In figure below an example is shown SNMP manager, in which some values are read.

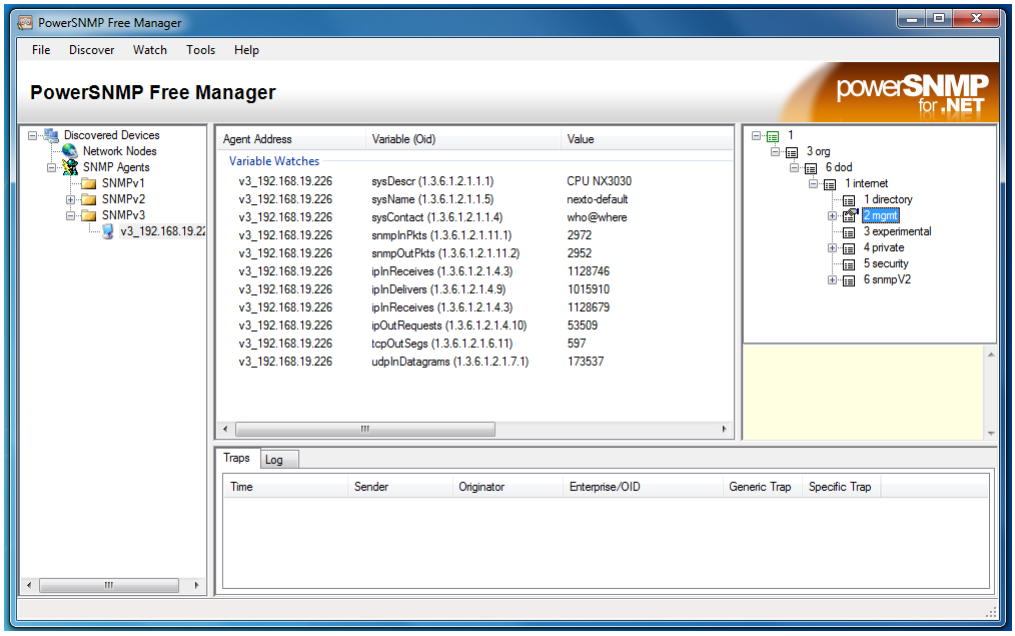


Figure 156: SNMP Manager Example

For SNMPv3, in which there is user authentication and password to requests via SNMP protocol, is provided a standard user described in the User and SNMP Communities section.

If you want to disable the service, change the SNMPv3 user or communities for SNMPv1 / v2c predefined, you must access the System Web Page of the CPU. For details, see the SNMP Configuration section.

5.13.3. Private MIB

The Private MIB has been discontinued since June 2019.

5.13.4. SNMP Configuration

SNMP settings can be changed through the System Web Page, in the *CPU Management* tab in the *SNMP* menu. After successful login, the current state of the service (enabled or disabled) as well as the user information SNMPv3 and communities for SNMPv1 / v2c can be viewed.

The user can enable or disable the service via a checkbox at the top of the screen.

It's also possible to change the SNMPv3 information by clicking the *Change* button just below the user information. Will open a form where you must complete the old username and password, and the new username and password. The other user information SNMPv3 cannot be changed.

To change the data of SNMPv1/v2c communities, the process is similar, just click the *Change* button below the information community. A new screen will open where the new data to the *rocommunity* and *rwcommunity* fields will be inserted. If you fail any of the fields blank, their community will be disabled. That way, if the user leaves the two fields blank, access to the SNMP agent will only be possible through SNMPv3.

If the user wants to return to the default settings, it must be manually reconfigure the same according to the User and SNMP Communities section. Therefore, all current SNMP configurations will be kept in the firmware update process. These options are shown in figure below.

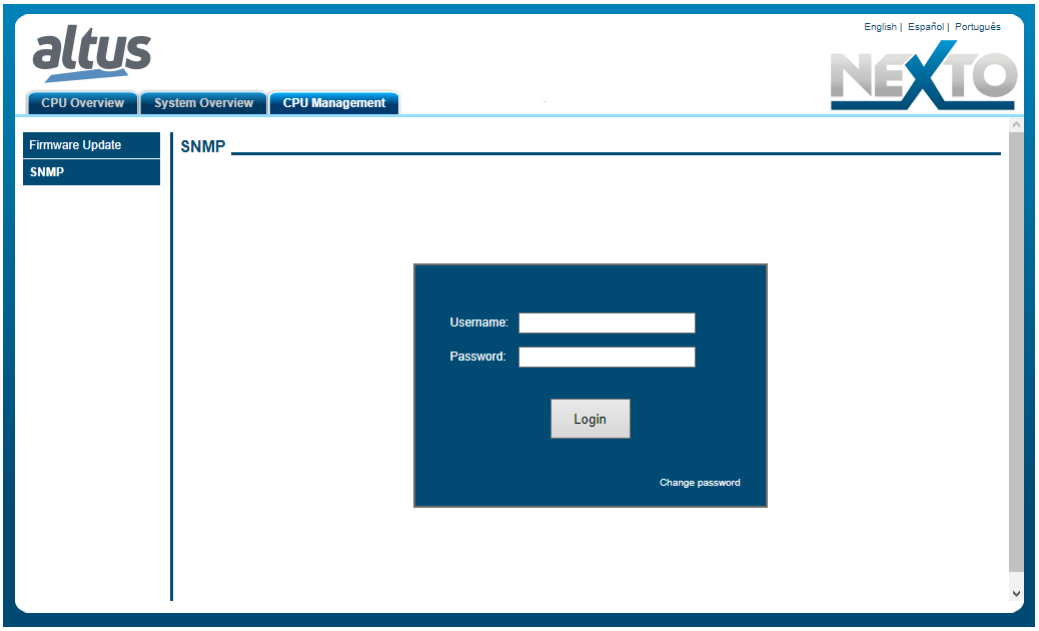


Figure 157: SNMP Login screen

After successful login, the current state of the service (enabled or disabled) as well as the user information SNMPv3 and communities for SNMPv1 / v2c can be viewed.

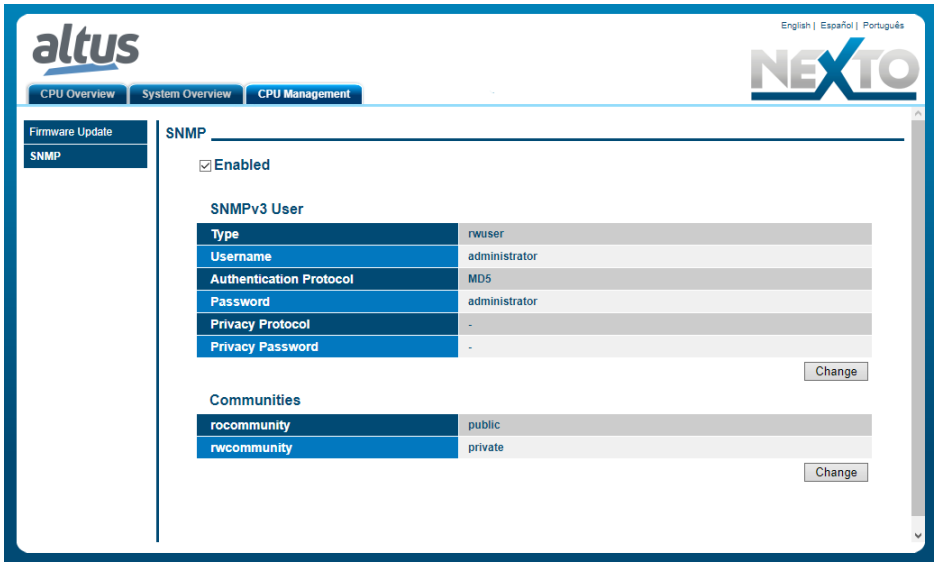


Figure 158: SNMP status configuration screen

ATTENTION

The *Username* and *Password* to access the agent via SNMP protocol are the same used to login on the SNMP Settings web page.

5.13.5. User and SNMP Communities

To access the SNMPv1 / v2c of the Nexto Series CPUs, there are two communities, according to table below.

Communities	Default String	Type
rocommunity	Public	Only read
rwcommunity	Private	Read and Write

Table 198: SNMPv1/v2c Default Communities info

It's possible to access SNMPv3 using default user, see table below:

Username	Type	Authentication Protocol	Password	Privacy Protocol	Privacy Password
administrator	rwuser	MD5	administrator	-	-

Table 199: SNMPv3 Default User info

For all settings of communities, user and password, some limits must be respected, as described on the following table:

Configurable item	Minimum Size	Max Size	Allowed Characters
rocommunity	-	30	[0-9][a-z][A-Z]@$*_.
rwcommunity	-	30	[0-9][a-z][A-Z]@$*_.
V3 User	-	30	[0-9][a-z][A-Z]@$*_.
V3 Password	8	30	[0-9][a-z][A-Z]@$*_.

Table 200: SNMP settings limits

6. Redundancy with NX3030 CPU

6.1. Introduction

This chapter describes the Nexto Series CPUs redundancy which can only be used with the NX3030 CPU.

Nexto's redundancy is of the hot-standby type, thus, the controllers are doubled. One controller is normally in active state and controlling a process, while the other is normally in stand-by state, keeping the synchronism with the active controller. In case of a failure in the active controller damaging its process control, the stand-by controller switches automatically to Active, within a very short time, in order not to disturb the process and cause any discontinuities in its outputs.

The hot-standby redundancy is a method used to increase failure tolerance and, consequently, increase the availability of automation systems. The basic idea is to ensure that no simple failure in duplicated components causes the process control interruption.

The hot-standby redundancy is applied on:

- Oil exploration platforms
- Energy generation and distribution plants
- Security interlock (Instrumented Security Systems)
- Continuous processes such as chemical plants, oil refinery, paper production, etc.

In the Nexto Series CPUs hot-standby redundancy, as it has already been described, the controllers are doubled. Besides, the field buses (PROFIBUS-DP) can also, optionally, be doubled, as well as the Ethernet supervisory networks and the Ethernet HSDN (*High Speed Deterministic Network*) control networks. By choosing the networks duplication, the availability becomes even higher.

The Nexto Series CPUs hot-standby redundancy is not applied to I/O modules. In case the I/O module redundancy is desired, it can be treated by the user in the application level. For instance, the user can duplicate or even triplicate an analog input module and create a vote scheme to define which input will be considered in an application specific time.

The figure below shows a typical example of redundant architecture using the NX3030 CPU.

The redundant CPU central part is formed by two identical racks, called PLCA and PLCB, and a redundancy control panel (PX2612). In the redundancy context, each rack (PLCA or PLCB) is called half-cluster, while the group formed by these two racks is called cluster.

In this example, a PROFIBUS network, Ethernet supervision network and Ethernet HSDN control network duplication can also be observed.

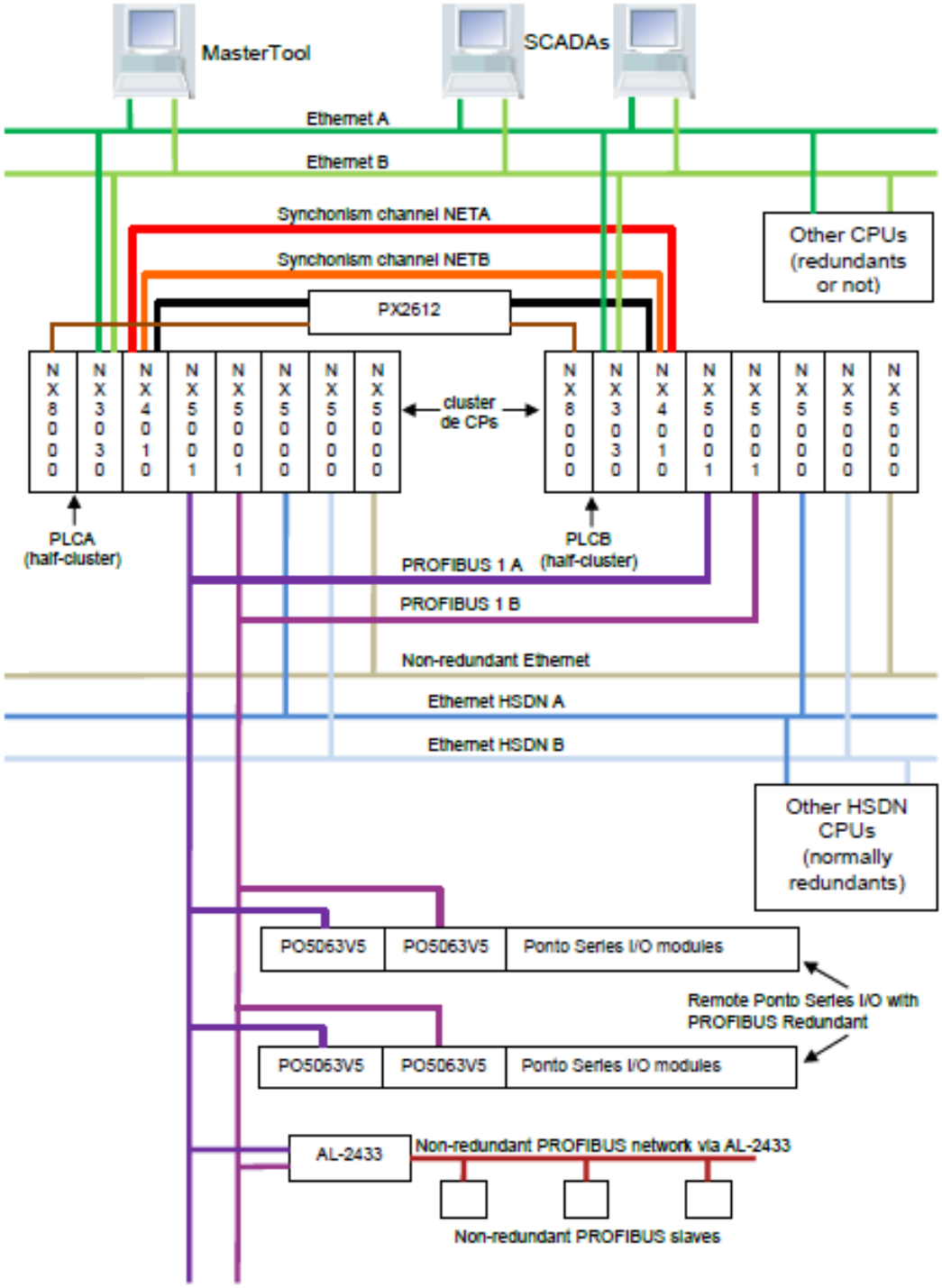


Figure 159: Example of redundant architecture with NX3030 CPU

6.2. Technical Description and Configuration

6.2.1. Minimum Configuration of a Redundant CPU (Not Using PX2612 Panel)

A redundant CPU is composed, at least, by:

- Two identical half-clusters

Each half-cluster consists of at least the following modules:

- The rack itself where the modules are inserted, which can be one of the following:
  - NX9000, with 8 positions
  - NX9001, with 12 positions
  - NX9002, with 16 positions
  - NX9003, with 24 positions
- The power supply NX8000, at rack positions 0 and 1
- The NX3030 CPU, at rack positions 2 and 3
- The module NX4010, at rack positions 4 and 5

The figure below shows an example of a redundant CPU minimum configuration, using the smallest rack (NX9001, with 12 positions). In this case, it can be observed that the three modules inserted in the rack have double width (occupy two rack positions).

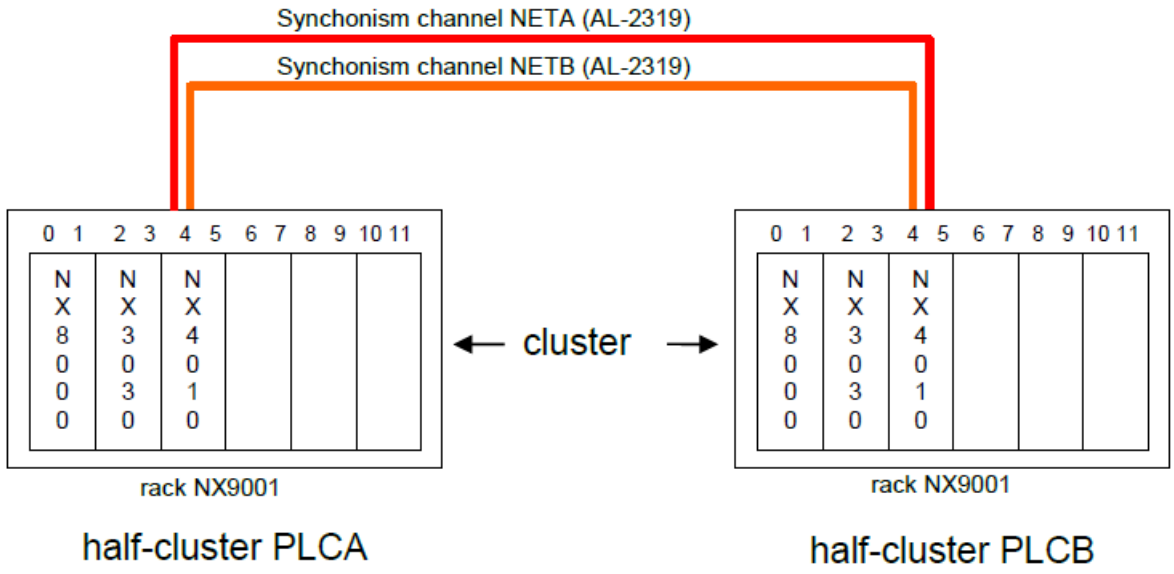


Figure 160: Minimum configuration of a redundant CPU in rack NX9001

6.2.2. Typical Configurations of a Redundant CPU

A minimum configuration, as the one shown on Figure 160, may already work as "redundant processing unit". It would be possible to use the serial and Ethernet communication channels from NX3030 CPU, for instance, for MODBUS TCP communication with a SCADA system, and MODBUS RTU and/or MODBUS TCP communication with smart field devices or MODBUS remote I/O systems.

In typical configurations, however, additional modules are inserted in the PLCA and PLCB half-clusters, for instance, to deliver a remote PROFIBUS I/O and Ethernet additional channels.

Among the additional modules which, optionally, may be inserted in the half-clusters, are the following:

- PROFIBUS Masters NX5001
- Ethernet Interfaces NX5000

In case is necessary, bigger racks can be used, as the NX9002 (16 positions) and NX9003 (24 positions). It must be observed that all the listed modules, so far in this chapter, have double width (occupy two positions). Additionally, it's also possible to use the PX2612 panel, which allows the execution of some redundant state machine transitions that, otherwise, wouldn't be possible, in addition to the automatic half-clusters shutdown in failure situations.

6.2.2.1. NX5001 Modules Addition for PROFIBUS Networks

A redundant PLC is up to until four NX5001 modules for PROFIBUS networks usage. Each network can be single or redundant. In case the PROFIBUS "n" (being "n" a number between 1 and 4) be redundant, the two networks that belongs to this are named PROFIBUS "n" A and PROFIBUS "n" B. In case the PROFIBUS "n" be single, the network that belongs to it will be named PROFIBUS "n" A.

To create a redundant PROFIBUS network, must be inserted two NX5001 modules in each half-cluster. To create a simple PROFIBUS network, simply insert a NX5001 module in each half-cluster. Thus, it can be configured up to four simple networks, or two redundant networks, or a redundant and two simple. In other cases, fewer than four NX5001 modules will be needed in each half-cluster. More information about PROFIBUS networks is provided in the PROFIBUS Network Configuration section.

In Figure 159, there is only one PROFIBUS network (PROFIBUS 1), and the same is redundant (PROFIBUS 1 A and PROFIBUS 1 B). In this example, therefore, were inserted two NX5001 modules in each half-cluster.

6.2.2.2. NX5000 Modules Addition for Ethernet Networks

It's possible to insert up to six NX5000 modules in each half-cluster, delivering six additional Ethernet channels, besides the two Ethernet channels already existent in the NX3030 CPU.

The Ethernet channels can be used in an individual way, or organized in NIC Teaming pairs, which are used to deliver redundant Ethernet channels, and are described, with more details, in the Redundant Ethernet Networks with NIC Teaming section.

A typical application for the NX5000 module can be the construction of a redundant HSDN (High Speed Deterministic Network), for the communication between several redundant CPUs. Through this network, many redundant CPUs can exchange messages in a totally segregated network, in order to guarantee determinism and a fast communication. Furthermore, configuring this network as redundant with NIC Teaming pairs, an excellent availability may be reached. In order to build such network (redundant HSDN), two NX5000 modules must be inserted in each half-cluster. Figure 159 shows a redundant HSDN example using two NX5000 modules in each half-cluster.

Applications where input and output modules are connected to Ethernet networks may require extra interface modules NX5000 to connect to these networks. In these cases, the network that connects the modules of inputs and outputs can be a simple or redundant network. Furthermore, the interfaces can be configured with the option of generating life failure. In this case, a network failure will cause a switch-over.

Figure 159 also shows an example with a NX5000 module used in the isolated form (without NIC Teaming redundancy), inserted at the right side from the other modules in each rack.

6.2.3. NX4010 Module

The NX4010 model, as shows figure below, was conceived in order to provide the interconnection between the two PLCA and PLCB half-clusters, and also to connect these half-clusters to the redundancy control panel PX2612. For further information regarding this module connections, see Interconnections between Half-Clusters and the Redundancy Control Panel PX2612.

Figure 161: NX4010

6.2.3.1. NX4010 Features

Its main features are:

- Data and application synchronization between two half-clusters
- Redundant communication interface between two half-clusters
- Automatic switchover (active half-cluster change) in case of NX4010 and CPU communication time-out
- Possibility to switch off the other half-cluster
- One Touch Diag
- Electronic Tag on Display
- Display and LEDs for diagnostics indication

Other features (generals, electrical, mechanic and environment) are presented in the NX4010 Redundancy Module Technical Characteristics - CE114900.

6.2.4. Redundancy Control Panel PX2612

The PX2612 control panel is an optional item in a redundant system. It must be used when the 'redundancy with panel' option is selected during the project creation using the wizard. Figure 162 shows the redundancy control panel, while Figure 163 shows the frontal panel with details.

Through the DB9 connector called CONTROL PLC A, the connection with the CONTROL connector from PLCA NX4010 is made, using the AL-2317/A cable.

Through the DB9 connector called CONTROL PLC B, the connection with the CONTROL connector from PLCB NX4010 is made, using the AL-2317/B cable.

Furthermore, there's a connector with 5 male terminals:

- GND: terminal for ground connection.
- RL A: 2 terminals connected to a relay NO (normally open) contacts, which can be commanded by PLCB to switch off PLCA. This relay must be closed by PLCB in order to switch off PLCA.
- RL B: 2 terminals connected to a relay NO (normally open) contacts, which can be commanded by PLCA to switch off PLCB. This relay must be closed by PLCA in order to switch off PLCB.

A CPU (PLCA or PLCB) can turn off the other CPU (PLCB or PLCA) in some exceptional situations, using the NO relays in the RLA and RLB connectors. Such situations are described in the Transition between Redundancy States section.

The PX2612 has also 6 buttons for redundancy command and 6 LEDs used for redundancy state indication. Each CPU reads 3 from these 6 buttons and controls 3 LEDs.

For further information regarding these buttons and LEDs functions, see PX2612 Redundancy Command Panel Functions section.

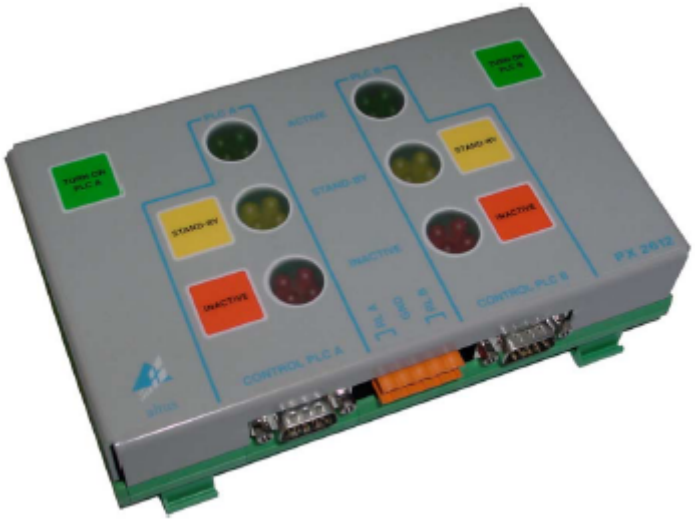


Figure 162: Redundancy Control Panel PX2612

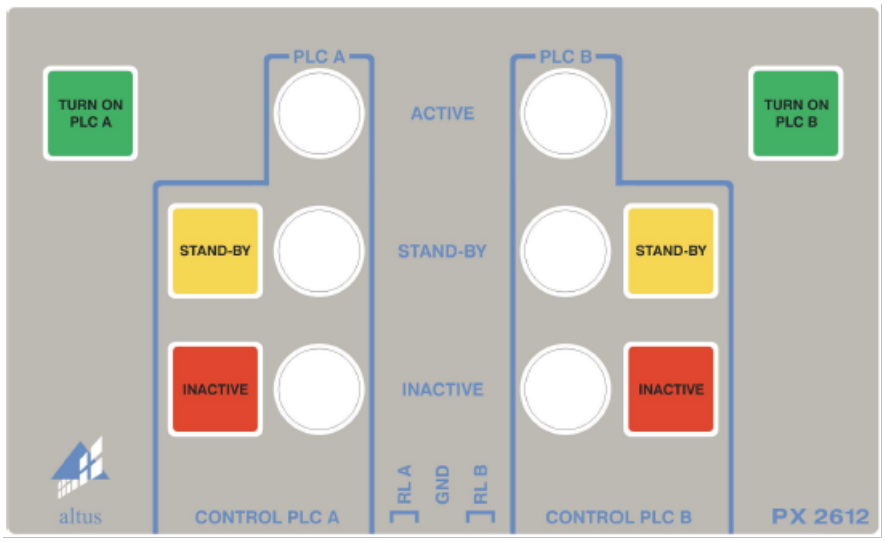


Figure 163: Redundancy Control Panel PX2612 Frontal View

6.2.4.1. PX2612 Features

The redundancy control panel PX2612 has the following features:

- CONTROL PLC A: connection to the module NX4010 from PLCA
- CONTROL PLC B: connection to the module NX4010 from PLCB
- RL A: relay NO terminals used to switch off PLCA
- RL B: relay NO terminals used to switch off PLCB

- GND: grounding

Other features (generals, electrical, mechanic and environment) are presented in the Redundancy Control Panel PX2612 Technical Characteristics - CT112500.

6.2.5. Interconnections between Half-Clusters and the Redundancy Control Panel PX2612

The figure below shows how the connections between PLCA, PLCB and PX2612 have to be made, including the possibility to allow a CPU to switch off the other, which is necessary in exceptional situations.

Two AL-2319 cables must be used for the synchronism and redundancy channels NETA and NETB. One of these two cables interconnects the NX4010 NET 1 connector from each CPU (NETA channel). The other cable interconnects the NX4010 NET 2 connector from each CPU (NETB channel).

An AL-2317/A cable interconnects the NX4010 CONTROL connector from the PLCA to the PX2612 CONTROL PLC A.

An AL-2317/B cable interconnects the NX4010 CONTROL connector from the PLCB to the PX2612 CONTROL PLC B.

Besides this, it's necessary to build a special power supply circuit in order to allow a CPU to switch off the other in extreme cases.

For higher reliability, two separate 24 V power supplies must be used, one for PLCA supply and other for PLCB supply.

It can be observed that is necessary to use two external relays from the normally closed type (NC), with current capacity to feed the NX8000. These relays must be dimensioned for a nominal current around 2 A, however, a current inrush of around 10 A must be taken into account. Shunt diodes connected to the NC relays solenoids must be used to protect the PX2612 NO relay contacts.

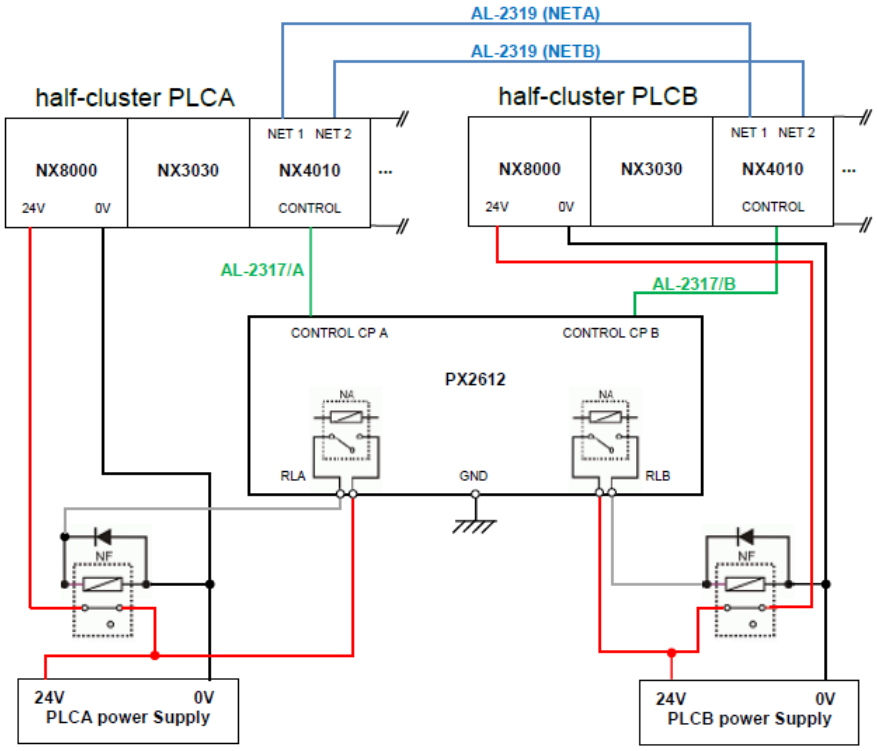


Figure 164: Interconnections between PLCA, PLCB and PX2612

6.2.6. General Characteristics of a Redundant CP

	Redundant CPU General Features
Allowed CPUs	NX3030
Redundancy type	Hot-standby
Failure tolerances	Tolerates, at least, simple failures in doubled equipment in the half-clusters. In specific cases, it can tolerate multiple failures.
Half-cluster 5 redundancy states	- Not-configured: initial state, also considered when the CPU is off or isn't executing the MainTask. - Starting: temporary state assumed after Not-configured, where some tests will define the next state (Inactive, Active, Stand-by or back to Not-configured. - Inactive: state reached after some types of failures or for programming maintenance. - Active: controlling the user process. - Stand-by: ready to switch to Active and control the user process, in case there's such demand (e.g. Active CPU failure).
Main failures which cause switchover between the Active CPU and the Reserve CPU. The reserve CPU switches for the Active and the Active can go to Inactive or Not-configured.	- Supplying failure. - Power supply. - CPU (stop in the MainTask execution). - NX4010. - Failure in both synchronism channels (NETA and NETB) and the cause isn't in the Reserve CPU. In this case the Reserve CPU, besides assuming the Active state, switches the other CPU off. - Failure of some synchronism channel (NETA and NETB) and the cause is in the Active CPU. - Failure in some vital PROFIBUS network. - Failure in some vital Ethernet network.
Commands that cause switchover between the Active CPU and the Reserve CPU	- Commands via redundancy control panel (PX2612). - Commands received from MasterTool or from a SCADA system, through this CPU (local) or the other CPU (remote). - Commands generated by user application (e.g.: in case of other diagnostics as Ethernet communication failure) through this CPU (local) or the other CPU (remote).
Main failures which prevents a CPU to go to the reserve state or remain in it. Such failures drive the CPU to a Not–Configured or Inactive state.	- Supplying failure. - Power supply. - CPU (stop in the MainTask execution). - NX4010. - Failure in both synchronism channels (NETA and NETB) and the cause is in the Reserve CPU. - Failure in the synchronism service for redundancy data. - Failure in the synchronism service for the redundant forcing list. - Total failure in some vital PROFIBUS network. - Total failure in some vital Ethernet network. - Different project from the Active CPU, with project automatic synchronization enabled. - Firmware version incompatible with the Active CPU.

	Redundant CPU General Features
Commands that drive the CPU out of the reserve state	- Commands via redundancy control panel (PX2612). - Commands received from MasterTool or from a SCADA system, through this CPU (local) or the other CPU (remote). - Commands generated by user application (e.g.: in case of other diagnostics as Ethernet communication failure) through this CPU (local) or the other CPU (remote).
Switchover time	- Up to 3 cycles from the MainTask, depending on the stimulus for state change (command or failure). - In case of PROFIBUS network failure, 2 MainTask cycles + 500 ms.
No discontinuities switchover (bump-less)	- A switchover doesn't cause discontinuities in the controller outputs, nor in the inner variables.
Redundancy overhead (MainTask cycle CPU consuming increased by redundancy)	- Maximum value automatically calculated by MasterTool and informed to the user, considering an empty redundant forcing list. - Typical average value of 60 ms for 224 kbytes of redundant data, in a system with a redundant PROFIBUS network and two redundant Ethernet HSDN networks
CPU display	Among other diagnostics, shows the redundancy state (Active, Stand-by, Inactive, Not-configured and Starting) together with the CPU identification PLCA or PLCB.
Redundancy Control Panel PX2612	- Through buttons, allows commands of switchover or redundancy states transition for maintenance. - LEDs signalize the redundancy state in each half-cluster. - NO relay allows a half-cluster to switch off the other in extreme situations. A button allows the other half-cluster reactivating. - Embedded resources for buttons, LEDs and relays tests. - A project created with panel cannot be converted to a project without panel, and vice-versa.
Redundancy diagnostics	- Indicate failures in the PLCA and in the PLCB, independent of their states (Active or Inactive). - Prevent "obscure failures". - Allow quick maintenance, essential for high availability.
Redundancy commands	- Allow the execution of the same PX2612 control panel actions, among other commands (e.g. switchover command). - Can be executed in the local CPU, or transmitted to the other CPU (remote) via synchronism channels NETA/NETB. - Can be received through MasterTool or a SCADA system. - Can be executed through user application.
Redundancy events	Register diagnostics and redundancy commands changes, with timestamp, allowing an investigation of the switchover causes.
SNTP (Simple Network Time Protocol)	Allow the events to have a precise timestamp adjusted to the world hour. It also synchronizes the CPU real time clock for other applications.
Commands and diagnostics synchronization	Each MainTask cycle, PLCA and PLCB exchange diagnostics and commands through synchronism channels NETA or NETB. This way, a CPU knows the other diagnostics and commands.

	Redundant CPU General Features
Redundant data synchronization	Each MainTask cycle, the Active CPU copies redundant data to the Inactive CPU through the synchronism channels NETA and NETB. Non-redundant data are not synchronized.
Redundant forcing list synchronization	Each MainTask cycle, the Active CPU copies the redundant forcing list to the Inactive CPU through the synchronism channels NETA and NETB. This list includes only forced redundant variables, this way PLCA and PLCB can have different non-redundant data groups forced, as these variables are not synchronized.
Single project for PLCA and PLCB	There's a single project for the PLCA and PLCB, generated by MasterTool. The project is composed by the application project (executable code) and the archive project (source code).
CPU identification	Through MasterTool, a NX3030 CPU identifies itself as PLCA, PLCB or non-redundant CPU. This identification isn't part of the application project generated by MasterTool, even though is written in a CPU using MasterTool. The CPU identification allows the feature of a single project for PLCA and PLCB.
Automatic project synchronization	If the Active CPU project becomes different from the Inactive CPU, it is copied from the first to the second. This synchronization can take several MainTask cycles. One must remember the project is composed by the application project (executable code) and the archive project (source code), and both are synchronized. This synchronization can be disabled in special cases in order to allow visualization of project modifications which can only be downloaded offline in non-redundant CPUs.
Online expansion of modules and PROFIBUS remotes	There are project modifications that can't be done online in a non-redundant CPU, such as the inclusion of new modules or PROFIBUS remotes. However, using the CPU and the PROFIBUS network redundancy, it was defined a procedure to accomplish this goal, very important for systems which need high availability.
Private IP addresses for PLCA and PLCB	It's possible to connect to a specific CPU (PLCA or PLCB) using a private IP address, to obtain half-cluster specific diagnostics, for instance. The PLCA IP address will always be associated to the PLCA NET(i) interface, while the PLCB IP address will always be associated to the PLCB NET(i) interface.
Active IP	Name of a strategy that allows the Ethernet client connect to a server from the redundant CPU using always the same IP address. This prevents the necessity of complex scripts to change the IP address when switchovers occur due to redundancy. The Active IP address will always be associated to the NET(i) interface from the Active CPU.
NIC Teaming	Name of the strategy which allows two Ethernet interfaces from a half-cluster to form a redundant pair sharing a same IP address. This way, redundant Ethernet network can be built easily, without the need for the clients, connected to a NIC Teaming, to implement complex scripts to switch IP addresses.

	Redundant CPU General Features
PROFIBUS Network and Vital Failures Configuration	The CPU supports up to 4 simple PROFIBUS networks or up to 2 redundant PROFIBUS networks. It's also possible to configure if each PROFIBUS network failure is considered vital (causes switchover) or not.
Ethernet Network and Vital Failures Configuration	The CPU supports up to 8 simple Ethernet networks or up to 4 redundant Ethernet networks (considering the NX3030 front interfaces). It's also possible to configure if each Ethernet network failure is considered vital (causes switchover) or not.
Single and cyclic user task	Only one user task is allowed, called MainTask. This task is cyclic.
Main POU programs: NonSkippedPrg and ActivePrg	At a redundant project creation, MasterTool generates automatically two empty POU programs, which must be filled by the user: - NonSkippedPrg: this POU is executed in both CPUs (PLCA and PLCB), independent on the redundancy state (Active or Inactive). It's used for diagnostics and special commands management. - ActivePrg: this POU is executed only in the Active CPU and is used for the final user's process control.

Table 201: General features of a redundant CPU

6.2.7. Purchase Data

The minimum configuration for a redundant CPU implies on the purchase of the following modules:

- Two racks, which must be chosen between the four available models according to the modules to be installed:
  - NX9000: 8 positions (4 double modules)
  - NX9001: 12 positions (6 double modules)
  - NX9002: 16 positions (8 double modules)
  - NX9003: 24 positions (12 double modules)
- Two NX8000
- Two NX3030
- Two NX4010
- Two AL-2319

Furthermore, it may be necessary to purchase the following additional modules:

- One PX2612
- One AL-2317/A
- One AL-2317/B
- Two modules NX5001 for each simple PROFIBUS network
- Four modules NX5001 for each redundant PROFIBUS network
- Two modules NX5000 for each additional simple Ethernet network
- Four modules NX5000 for each additional redundant Ethernet network (NIC Teaming)

ATTENTION

It can be installed up to 4 PROFIBUS modules in each half-cluster. This means that we can configure up to 4 simple PROFIBUS networks or up to 2 redundant PROFIBUS networks.

6.3. Principles of Operation

In this section, the redundant CPU functions using a NX3030 CPU is described, along with its behavior and states. It's also presented concepts and programming and configuration restrictions that will be used in the next sections.

6.3.1. Identification of an NX3030 CPU

A NX3030 CPU has a non-volatile identification register where it's possible for it to be identified as:

- Non-redundant: it can't be used in a redundant CPU (default configuration)
- PLCA: used in the redundant CPU PLCA
- PLCB: used in the redundant CPU PLCB

The identification register can be adjusted using the MasterTool programmer. The first thing to be done in a NX3030 CPU, before downloading the redundant project in it, is to identify it as PLCA or PLCB. In case the identification isn't executed, several redundancy features won't work correctly, as, for instance, the synchronization of the application between the PLCs.

ATTENTION

The CPU identification register is not part of the redundant CPU project, thus it isn't saved as part of this project in the computer where MasterTool is being executed. The register is saved only in the non-volatile CPU memory.

6.3.2. Single Redundant Project

Due to the identification register previously described, there's a single project for the redundant CPU, identical for both PLCA and PLCB.

Configuration parameters that must be different for PLCA and PLCB (e.g. Ethernet interface IP address) appear doubled in the redundant CPU project (one for the PLCA and another for the PLCB). Each CPU will consider the correspondent one, after analyzing its identification register.

6.3.3. Redundant Project Structure

6.3.3.1. Redundancy Template

A redundant CPU project is created automatically from a model, called Redundancy Template.

The template starts from the minimum redundant CPU configuration, as defined in the Minimum Configuration of a Redundant CPU (Not Using PX2612 Panel) section. Besides this, some dialogs with the user are made for the insertion of additional modules in the half-clusters, such as PROFIBUS masters (NX5001) and Ethernet modules (NX5000).

PROFIBUS remotes must be inserted by the user, below the NX5001 PROFIBUS masters already inserted.

Furthermore, tasks and basic POUs from the program type are created, as described in the following sections.

6.3.3.2. Single and Cyclic Task MainTask

The redundant CPU project has a single task, called MainTask, which is cyclic. The user can adjust the task cycle time.

6.3.3.3. MainPrg Program

The MainTask is connected to a single POU from the program type, called MainPrg. The MainPrg program is created automatically.

The MainPrg code is the following, in ST language:

```
SpecialVariablesPrg();
IF isFirstCycle THEN
   StartPrg();
   isFirstCycle := FALSE;
ELSE
   fbRedundancyManagement();
```

```
    IF fbRedundancyManagement.m_fbDiagnosticsLocal.eRedState = REDUNDANCY_STATE.
    ACTIVE THEN
       SpecialVariablesRedundantPrg();
    END_IF;
    NonSkippedPrg();

    IF fbRedundancyManagement.m_fbDiagnosticsLocal.eRedState = REDUNDANCY_STATE.
    ACTIVE THEN
       ActivePrg();
    END_IF;
END_IF;
```

MainPrg call two POUs from the program type, called NonSkippedPrg and ActivePrg. NonSkippedPrg is always called, as it's executed in both CPUs. On the other hand, ActivePrg is only called when the "RedDgnLoc.sGeneral.Diag.eRedState = Active" condition is true, in other words, when the CPU is in active state.

However, the NonSkippedPrg program is executed in both CPUs (PLCA and PLCB) independent on the redundancy state of this CPU. On the other hand, the ActivePrg is executed only in the active CPU.

Opposite to the MainPrg, which must not be modified, the user may modify the NonSkippedPrg and ActivePrg programs. Initially, when the redundant project is created from the Redundancy Template, these two programs are created "empty", but after that the user may insert his code.

ATTENTION

When the OPC option is enabled when creating the project, the NonSkippedPrg program is not created empty. For more information, refer to the OPC DA Communication Use with Redundant Projects section.

6.3.3.4. ActivePrg Program

The main goal of this program, which is executed only in the active CPU, is to control the final user process.

This program normally acts on the redundant variables, among which the direct representation variables are found %I and %Q associated to the remote I/O system. For further information see the section Redundant CPU Programming - MainTask Configuration - ActivePrg Program.

ATTENTION

The compilation being successful or not, MasterTool informs the calculated looseness and the redundancy overhead predicted on the message window.

6.3.3.5. NonSkippedPrg Program

This program is executed in both CPUs (PLCA and PLCB) independent on the redundancy state. It's typically used for functions such as:

- To organize non-redundant diagnostics to report to a SCADA system.
- To receive and execute non-redundant commands from a SCADA system.
- To manage switchover conditions normally not automatically contemplated by the redundant CPU, that can vary from user to user. E.g. a user will be able to execute a switchover to the Reserve CPU if the Active CPU isn't communicating with the SCADA system, while another user may not want a switchover on this situation.
- To enable or disable I/O drivers according to the redundancy state, e.g. disable a Modbus RS-485 master in the Inactive CPU.
- To detect failures in I/O drivers in an inactive CPU, in order to avoid obscure failures. Some I/O drivers don't include such failures automatically detection, while others, such as the PROFIBUS, does it automatically.
- Other activities which, for some reason, need to be executed either in the Active CPU and the Reserve CPU.

For further information see Redundant CPU Programming - MainTask Configuration - NonSkippedPrg Program section.

6.3.3.6. Redundant and Non-redundant Variables

The redundant CPU variables can be classified among redundant and non-redundant. Redundant variables are copied from the Active CPU to the Inactive CPU, at the MainTask beginning of each cycle, through the synchronism channels NETA and NETB. On the other hand, non-redundant variables aren't copied between half-clusters, thus can have different values in PLCA and PLCB.

The non-redundant variables are used to store private information of each half-cluster (PLCA or PLCB), such as module diagnostics inside the half-cluster, including the redundancy diagnostics (half-cluster diagnostics state, etc...).

The redundant variables regard the shared information connected to the process control. The variables associated to the I/O modules are typical examples of redundant variables.

6.3.3.7. Redundant and Non-redundant %I Variables

The NX3030 CPU allocates 96 kbytes of %I variables (%IB0 ... %IB98303).

The first 80 kbytes can be redundant (%IB0 ... %IB81919). The last 16 kbytes are always non-redundant (%IB81920 ... %IB98303).

The 80 kbytes area which can be redundant is allocated for inputs, which can be read from an I/O remote module (PROFIBUS, MODBUS, etc...).

The 16 kbytes non-redundant area is allocated for a half-cluster "quick private diagnostics", and also for the redundancy command panel PX2612 buttons. Quick diagnostics are the ones that must be updated each MainTask cycle.

The user may configure the redundant %I variables quantity, between 0 and 81920 kbytes, in 1 kbyte multiples (the default value is 16384 bytes - %IB0 ... %IB16383). The proper configuration of redundant %I from %IB0 is important to decrease the necessary time for redundant variables synchronization (decrease the redundancy overhead). E.g. if the real application allocates only %IB0 ... %IB1499 for redundant inputs, the redundant %I area size can be defined as 1500 bytes.

The figure below illustrates the redundant and non-redundant %I direct representation variables allocation, where RI is the %I quantity really configured as redundant.

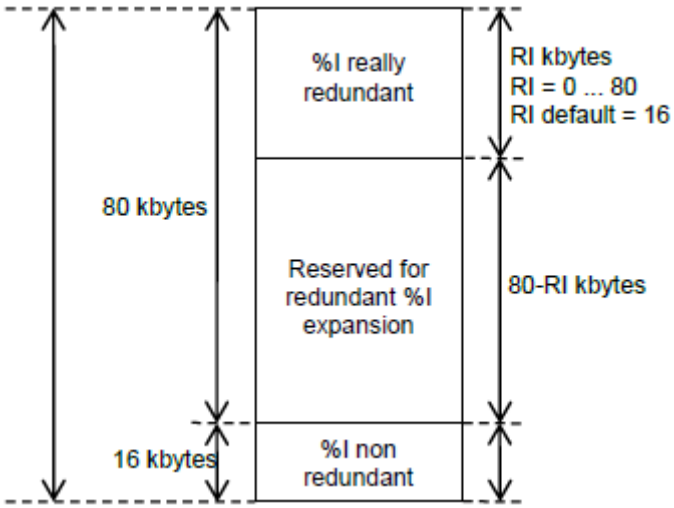


Figure 165: Redundant and Non-redundant %I Allocation

6.3.3.8. Redundant and Non-redundant %Q Variables

The NX3030 CPU allocates 96 kbytes of %Q variables (%QB0 ... %QB98303). The first 80 kbytes can be redundant (%QB0 ... %QB81919). The last 16 kbytes are always non-redundant (%QB81920 ... %QB98303).

The 80 kbytes area which can be redundant is divided in two sections:

- The first kbytes are reserved for outputs that can be redundant, and are typically allocated for an I/O remote system (PROFIBUS, MODBUS, etc.). The size value is configurable and its default value is 16384. By default, this section includes %QB0 ... %QB16383, and can be configured for up to 64 kbytes (%QB0 ... %QB65535).

- The next bytes are reserved for diagnostics which can be redundant, from the I/O system (I/O modules diagnostics, communication interfaces diagnostics, PROFIBUS slaves diagnostics, etc.), for instance. Different from the quick diagnostics (allocated in %I), such diagnostics allocated in %Q can take more than one MainTask cycle to be updated. By default this section includes 16 kbytes (%QB65536 ... %QB81919).

The non-redundant area (%QB81920 ... %QB98303) is typically allocated for diagnostics and private commands of a half-cluster, and also for the redundancy command panel PX2612 LEDs and relay.

The user can reduce the redundant %Q variable quantity in each one of the sections which can be redundant:

- On the first section, the really redundant area size can be configured between 0 bytes and 65535 bytes, in 1 byte multiples (the default value is 16384 bytes). The proper configuration of redundant %Q is important to decrease the necessary time for redundant variables synchronization (decrease the redundancy overhead). E.g. if the real application allocates only %Q0 ... %Q1499 for redundant outputs, the redundant %Q area size can be defined as 1500 bytes.
- On the second section, the really redundant area size can be configured between 0 bytes and 81919 bytes, in 1 byte multiples (the default value is 16384 bytes). The proper configuration of redundant %Q is important to decrease the necessary time for redundant variables synchronization (decrease the redundancy overhead). E.g. if the real application allocates only %QB65536 ... %QB66999 for redundant diagnostics, the redundant %Q area size can be defined as 1464 bytes.

The figure below illustrates the redundant and non-redundant %Q direct representation variables allocation, where RQS is the %Q output quantity configured as redundant in the first section, and RQD is the %Q diagnostics quantity configured as redundant in the second section.

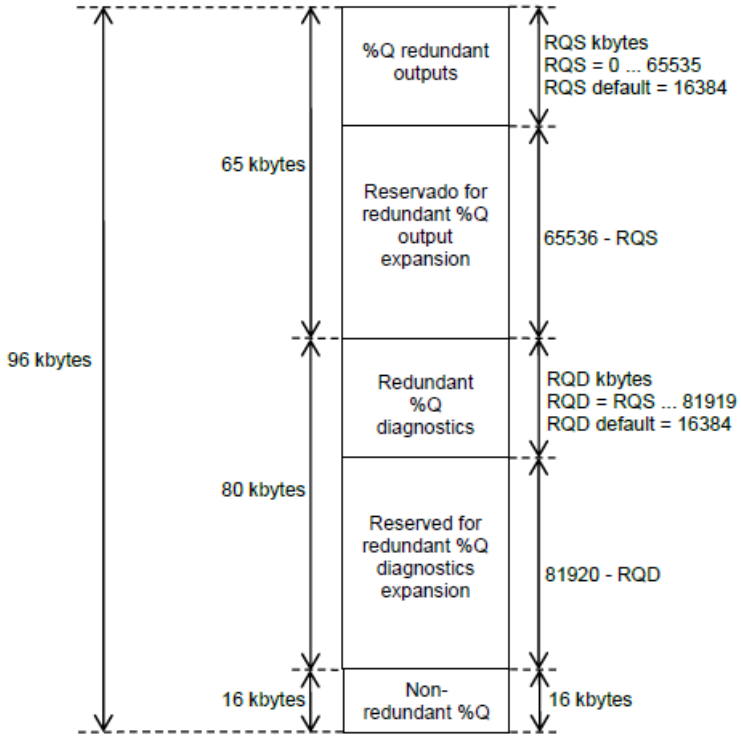


Figure 166: Redundant and Non-redundant %Q Allocation

6.3.3.9. Redundant and Non-redundant %M Variables

The NX3030 CPU allocates 64 kbytes of %M variables (%MB0 ... %MB65535).

All the 65535 bytes can be redundant (%MB0000 ... %MB65535). By default the redundant %M variables quantity is 0.

The %M variable use must be avoided and the use of symbolic variables preferred (see Redundant and Non-redundant Symbolic Variables section).

6.3.3.10. Redundant and Non-redundant Symbolic Variables

Besides the direct representation variables (%I, %Q and %M) which are allocated automatically, the user can explicitly declare symbolic variables, inside of POUs or GVLs. The maximum size allowed for redundant symbolic variables allocation is 512 kbytes.

ATTENTION

Symbolic variables must not be confused with AT variables. The AT variables are mere symbolic names attributed to direct representation variables (%I, %Q and %M), using the "AT" declaration. Thus, AT variables don't allocate any symbolic variables memory.

Symbolic variables are redundant in the following cases:

- When declared in POUs from the "program" type created in the user application, exceptionally the NonSkippedPrg program
- When declared in GVLs created in the user application and these GVLs marked as redundant

Symbolic variables aren't redundant in the following cases:

- When declared in the NonSkippedPrg program. This program has been described previously in the NonSkippedPrg Program section
- When declared in POUs from the "function" type. It can be observed this POUs normally must allocate variables only on the stack (non static), which consequently don't need to be redundant. If the user declares static variables (VAR STATIC) inside the POUs from the "function" type, this will be considered bad programming. Such static variables, in case they are created, will be considered non-redundant
- When declared in POUs from the "function block" type. It can be observed the mere "function block" declaration doesn't allocate memory (what allocates memory is to turn a function block into as instance)

It must be observed that the function blocks instances, declared inside POUs from the program type or inside GVLs, behave as symbolic variables, in other words, allocate redundant memory. In the same way symbolic variables, when function block instances, are declared in the NonSkippedPrg program or when the GVL isn't marked as redundant, such instances are non-redundant.

6.3.4. Multiple Mapping

If the user desires to map the redundant command variables in more than one communication port (COMx or NETx) it's necessary the implementation of a control by the user within his own application.

The control logic to be implemented must write in the redundant command variables based on the variables (commands) values from each communication port (COMx or NETx). Besides that, the control logic must restart the communication ports command variables, as the redundancy control just restarts its own command variables.

The following is an example of this implementation:

```
VAR
var_StandBy_command_Ethernet_relation : BOOL;
var_StandBy_command_Serial_relation : BOOL;
var_Inactive_command_Ethernet_relation : BOOL;
var_Inactive_command_Serial_relation : BOOL;
var_TurnOn_command_Ethernet_relation : BOOL;
var_Turn_command_Serial_relation : BOOL;
END_VAR

//Logic to put the local PLC in StandBy
IF var_StandBy_command_Ethernet_relation = TRUE THEN
DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal:=TRUE;
var_StandBy_command_Ethernet_relation:=FALSE;
END_IF
IF var_StandBy_command_Serial_relation = TRUE THEN
DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal:=TRUE;
var_StandBy_command_Serial_relation:=FALSE;
END_IF
```

```
//Logic to put the local PLC in Inactive
IF var_Inactive_command_Ethernet_relation = TRUE THEN
DG_NX4010.tRedundancy.RedCmdLoc.bInactiveLocal:=TRUE;
var_Inactive_command_Ethernet_relation:=FALSE;
END_IF
IF var_Inactive_command_Serial_relation = TRUE THEN
DG_NX4010.tRedundancy.RedCmdLoc.bInactiveLocal:=TRUE;
var_Inactive_command_Serial_relation:=FALSE;
END_IF
//Logic to switch on the local PLC switched off by the PX2612
IF var_TurnOn_command_Ethernet_relation = TRUE THEN
DG_NX4010.tRedundancy.RedCmdLoc.bTurnOnLocal:=TRUE;
var_TurnOn_command_Ethernet_relation:=FALSE;
END_IF
IF var_Turn_command_Serial_relation = TRUE THEN
DG_NX4010.tRedundancy.RedCmdLoc.bTurnOnLocal:=TRUE;
var_Turn_command_Serial_relation:=FALSE;
END_IF
```

Above there's an example in ST language, where the redundancy command can be executed through two variables from different communication ports. On the same example, three different commands were executed (StandBy, Inactive and TurnOn).

Where:

var_StandBy_command_Ethernet_relation: Bool type variable attributed to an Ethernet communication Coil which will execute the command to put the local Half-Cluster in Stand-By.

var_StandBy_command_Serial_relation: Bool type variable attributed to a Serial communication Coil which will execute the command to put the local Half-Cluster in Stand-By.

DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal: this command executes an action similar to the button STAND-BY from the PX2612, in the local PLC.

var_Inactive_command_Ethernet_relation: Bool type variable attributed to an Ethernet communication Coil which will execute the command to put the local Half-Cluster in Inactive.

var_Inactive_command_Serial_relation: Bool type variable attributed to a Serial communication Coil which will execute the command to put the local Half-Cluster in Inactive.

DG_NX4010.tRedundancy.RedCmdLoc.bInactiveLocal: this command executes an action similar to the button INACTIVE from the PX2612, in the local PLC.

var_TurnOn_command_Ethernet_relation: Bool type variable attributed to an Ethernet communication Coil which will execute the command to reactivate the local Half-Cluster after switched off by the PX2612 relay.

var_Turn_command_Serial_relation: Bool type variable attributed to a Serial communication Coil which will execute the command to reactivate the local Half-Cluster after switched off by the PX2612 relay.

DG_NX4010.tRedundancy.RedCmdLoc.bTurnOnLocal: this command executes an action similar to the button TURN ON PLC from the PX2612, in the local PLC.

6.3.5. Diagnostics, Commands and User Data Structure

Each CPU has several data structure related to redundancy. The following structure is AT variables mapped over %Q variables:

- RedDgnLoc: has diagnostics from the CPU (local) related to the redundancy, as the CPU redundancy state, for instance
- RedDgnRem: it's a copy from the other CPU RedDgnLoc, received through NETA/NETB synchronism channels. This way, this CPU (local) has access to the other CPU (remote) diagnostics
- RedCmdLoc: has commands which must be applied on this CPU (when called Local) or on the other CPU (when called Remote). E.g. the StandbyLocal field from this data structure corresponds to a command which must be executed in this CPU (local), while the StandbyRemote field corresponds to a command which must be executed in the other CPU (remote)
- RedCmdRem: it's a copy from the other CPU RedCmdLoc, received through NETA/NETB synchronism channels. This way, this CPU (local) can execute commands received from the other CPU (remote)

- RedUsrLoc: has 128 bytes of data filled freely by the user (e.g. communication diagnostics with a SCADA system). These 128 bytes of data can be interchanged with the other CPU (remote)
- RedUsrRem: it's a copy from the other CPU RedUsrLoc, received through NETA/NETB

On Redundancy Maintenance section, the following sub-sections offer more details regarding these data structures:

- Redundancy Diagnostics Structure
- Redundancy Commands
- User Information Exchanged between PLCA and PLCB

6.3.6. Cyclic Synchronization Services through NETA and NETB

This section describes the three synchronization services which occur cyclically in a redundant CPU between PLCA and PLCB, through NETA and NETB synchronism channels.

These services are executed at the beginning of each MainTask cycle, and in the sequence which they appear below:

- First, the Diagnostics and Commands Exchange service is executed
- Second, the Redundant Data Synchronization service is executed
- Third, the Redundant Forcing List Synchronization service is executed

6.3.6.1. Diagnostics and Commands Exchange

This service is responsible by the interchange of the following data structures, in each MainTask cycle:

- To copy RedDgnLoc from PLCA to PLCB RedDgnRem
- To copy RedCmdLoc from PLCA to PLCB RedCmdRem
- To copy RedUsrLoc from PLCA to PLCB RedUsrRem
- To copy RedDgnLoc from PLCB to PLCA RedDgnRem
- To copy RedCmdLoc from PLCB to PLCA RedCmdRem
- To copy RedUsrLoc from PLCB to PLCA RedUsrRem

The service will be executed using only one synchronism channel (NETA or NETB). This way the service can be completed even if one channel has problems.

6.3.6.2. Redundant Data Synchronization

This service is responsible for the redundant variables transferring, from the Active CPU to the Inactive CPU. As previously described, there are symbolic redundant variables and also redundant direct representation variables (%I, %M and %Q).

For this service to be executed, several conditions must be satisfied:

- The previous synchronization service in this MainTask cycle (Diagnostics and Commands Exchange) must be completed with success.
- In case this CPU is in Active state, the other must be in Non-Active state. On the other hand, in case this CPU is in Non-Active state, the other must be in Active state.
- Both projects (2 CPUs) must be identical, except when the project automatic synchronization is disabled (see Project Synchronization Disabling) section.
- At least one synchronism channel (NETA and/or NETB) must be operational. If both synchronism channels (NETA and NETB) are operational, the communication is distributed between both (load balances) in order to reduce the synchronization time. In case only one channel is operational, the synchronism will continue to be executed only by this channel, keeping the redundant data synchronization.

6.3.6.3. Redundant Forcing List Synchronization

This service is responsible for the redundant forcing list transferring, from the Active CPU to the Inactive CPU.

For this service to be executed, several conditions must be satisfied:

- Both synchronization services previous to this cycle (Diagnostics and Commands Exchange) must be completed with success
- In case this CPU is in Active state, the other must be in Non-Active state. On the other hand, in case this CPU is in Non-Active state, the other must be in Active state
- Both projects (2 CPUs) must be identical, except when the project automatic synchronization is disabled (see Project Synchronization Disabling) section
- At least one synchronism channel (NETA and/or NETB) must be operational. If both synchronism channels (NETA and NETB) are operational, the communication is distributed between both (load balances) in order to reduce the synchronization time. In case only one channel is operational, the synchronism will continue to be executed only by this channel, keeping the redundant data synchronization

ATTENTION

The redundant forcing list has only forcing over redundant variables. On each CPU (PLCA and PLCB), there can be a different forcing list related to non-redundant variables.

6.3.7. Sporadic Synchronization Services through NETA and NETB

The following synchronization services are executed sporadically, in other words, they are not executed in each MainTask cycle. Another system task executes these sporadic services in background.

6.3.7.1. Project Synchronization

This service is responsible for synchronizing the Active CPU and Non-Active CPU projects. This happens when the projects are different in both CPUs and the automatic projects synchronization is enabled on both CPUs.

The synchronization is always from the Active CPU to the Non-Active CPU and it's enough that one out of two synchronism channel (NETA or NETB) is operational for this service to be executed.

When the synchronization is enabled, the following files and services will be synchronized:

- Project application (executable code)
- Project archive (source code)
- Users and groups
- Access rights
- Trace

The synchronization service will start within thirty seconds after one of the CPUs goes to Active state, and after its beginning, the project CRC will be checked every five seconds.

When synchronization is started the Non-Active CPU goes to Stop mode, at the Not-Configured state. After the transferring of all necessary files, the Non-Active CPU goes to Run, at Starting state. In case the transfer fails, the CPU goes back to Not-Configured state.

The time the synchronization will take to be fully executed depends on the project size. In average, a transfer rate between the synchronism channels is approximately 500 kbytes/s.

In case the synchronization is interrupted (communication loss between synchronism channels) during the files transferring from the Active CPU to the Non-Active CPU, the procedure is aborted and restarted when the communication is restored. Only after the conclusion of the whole procedure the Non-Active CPU goes to Run mode.

Besides keeping the projects synchronized, the Project Synchronization will also avoid the Non-Active CPU to assume superior states in relation to Starting in case the CRC is different or some Online Change is to be executed in the Active CPU.

ATTENTION

A project synchronization will have the same effect as a download in the Non-Active CPU. This service isn't executed if the automatic Project Synchronization is disabled, as it's described on Project Synchronization Disabling section.
No synchronization service between PLCA and PLCB works in case the synchronism channels cables are inverted. E.g. to connect the NETA channel in the NETB channel from PLCB and the NETB from the PLCA in the NETA in the PLCB.

ATTENTION

In the update from the version 1.20 to later versions of MasterTool IEC XE, was done a modification in the communication protocol between the synchronism channels. Therefore, is not possible to sync data between two PLCs when one of the applications has been created in a version prior to 1.21 and another application has been created in an equal or higher version. To be able to perform the synchronization, you should perform the actions described at section Not Loading the Application at Startup of the PLC with the oldest project. Doing this, the application will not be loaded, but, when this PLC goes to Non-Configured state during the system initialization, the applications will be synchronized automatically.

ATTENTION

Before version 2.01 of MasterTool IEC XE, when sending the source code to the active CPU, the Stand-by CPU went for Not-Configured state to sync it. However, to complete the synchronization operation, the CPU remained in the state Not-Configured, being necessary to pass the CPU to Stand-by status via STAND-BY button on the PX2612 or equivalent command. Starting with version 2.01 the CPU that is in Stand-by will change your state to Not-Configured during the synchronization process, but will return automatically when the sources are the same between the two Half-Clusters.

6.3.8. Project Synchronization Disabling

On Sporadic Synchronization Services through NETA and NETB section, application project and archive project synchronization services were described. These services normally must be enabled, and are useful when the project modifications can be downloaded online in the Active CPU and the Stand-by CPU afterwards, automatically, through the synchronism channels NETA/NETB.

However, there are project modifications which can't be downloaded online in any CPU, e.g. the inclusion of modules in a PROFIBUS remote, or the inclusion of a new PROFIBUS remote. In these cases, using the CPU and PROFIBUS network redundancy, such modifications can be made without interrupting the process control. A procedure to accomplish this objective is described in the Exploring the Redundancy for Offline downloading of Modifications without Interruption of the Process control section.

In this procedure it's necessary to disable temporarily the project synchronizations, allowing, for a while, one CPU to operate with a project new version, while the other CPU still operates with the old project version.

A NX3030 CPU has a register for Project Synchronization Disabling, non-volatile, which allows the disabling of the project application and project archive synchronization services. This register can be adjusted using MasterTool. It's enough to disable the project synchronization in one of the two CPUs to guarantee it doesn't work anymore.

To disable the Project synchronization, the user must, firstly, connect into desired PLC with the software MasterTool (see section MasterTool Connection with a NX3030 CPU from a Redundant PLC).

Next, in the Online / Redundancy Configuration menu, the combo-box Project Synchronization must be opened, allowing the selection of the two following options:

- Enable
- Disable

The option "*Disable*" must be selected and the combo-box correspondent "*Write*" button pressed. A message informs if the operation is successful or not.

The disabling configuration of project synchronism isn't part of the redundant project developed in the MasterTool. Such configuration is only in a non-volatile memory area in the CPU, which can be read or written using MasterTool. MasterTool doesn't save this configuration in any file.

This configuration is copied on each cycle of MainTask, from the non-volatile memory to the DG_NX4010.tRedundancy.RedDgnLoc .sGeneral_Diag.bProjectSyncDisable. The user can verify this diagnostics in the PLC to see if the command succeeded, since the PLC is in Run mode (DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bProjectSyncDisable must be 1). In case the PLC isn't in Run mode, it's possible to verify configuration straight on the NX3030 CPU display in the PLC (see Redundancy Diagnostics on NX3030 CPU Graphic Display section).

The DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bProjectSyncDisable diagnostic can also be observed also in the remote PLC through the DG_NX4010.tRedundancy.RedDgnRem.sGeneral_Diag.bProjectSyncDisable (since the Non-Active PLC is in Run mode). A PLC (Active or Non-Active) stops the project synchronization service every time any of the following bits are true:

- DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bProjectSyncDisable
  - This PLC, local bit. This PLC is with the project synchronization disabled

- DG_NX4010.tRedundancy.RedDgnRem.sGeneral_Diag.bProjectSyncDisable
  - The other PLC, remote bit. The remote PLC is with the project synchronization disabled

ATTENTION

The Project Synchronization Disabling register isn't part from the redundant CPU project, thus it's not saved as part of it in the computer where MasterTool is being executed. The register is saved only in the non-volatile CPU memory.

6.3.9. PROFIBUS Network Configuration

It's possible to install up to four PROFIBUS Master NX5001 modules in each half-cluster. So, we can define up to two redundant PROFIBUS networks, called PROFIBUS 1 and PROFIBUS 2, or up to four simple PROFIBUS networks, called PROFIBUS 1, PROFIBUS 2, PROFIBUS 3 and PROFIBUS 4, or even one redundant network and two simple ones, named PROFIBUS 1, PROFIBUS 2 and PROFIBUS 3.

6.3.9.1. PROFIBUS Redundancy

Each of the PROFIBUS networks can be redundant or non-redundant. For example, if PROFIBUS 1 network is redundant, it will be divided into PROFIBUS 1 A and PROFIBUS 1 B. If it's non-redundant, there's going to exist only PROFIBUS 1 A. The same applies to the PROFIBUS 2.

Figure 159 shows an example with a single PROFIBUS network (PROFIBUS 1), which is redundant (PROFIBUS 1 A and PROFIBUS 1 B).

Only a few remote types can be connected straight to this redundant PROFIBUS network:

- PO5063V5: PROFIBUS slave DP-V0 for Ponto Series remotes
- PO5065: PROFIBUS slave DP-V1 with Hart, for Ponto Series remotes
- AL-3416: PROFIBUS slave DP-V0 for AL-2004 CPU
- NX5210: PROFIBUS slave DP-V0 for Nexto Series remotes

Figure 159 also shows the possibility to connect non-redundant remotes to this type of redundant PROFIBUS network, through the AL-2433 module (ProfiSwitch). Such non-redundant PROFIBUS remotes can be from any brand or model.

6.3.9.2. PROFIBUS Failure Modes Vital and Not-Vital

Each one of the PROFIBUS networks can be configured in two different modes:

- Vital failure: in case this network fails completely, this failure can determine a redundancy state transition in the redundant CPU (switchover). In case a redundant PROFIBUS network, a complete failure implies in the failure of both composing networks.
- Not-Vital failure: even if this network fails completely, this failure won't determine a redundancy state transition in the redundant CPU (switchover).

6.3.10. Redundant Ethernet Networks with NIC Teaming

Figure 159 shows two redundant Ethernet networks examples, with NIC Teaming.

In the first case, the NX3030 CPU connects to the supervision network (SCADA), also used for configuration through MasterTool. Both NX3030 CPU Ethernet ports (NET 1 and NET 2) form a NIC Teaming redundant pair, interconnected in two different switches (Ethernet A and Ethernet B). In some point, these two switches must be interconnected, for the two NIC Teaming ports connection and for an even higher availability (against double failures).

In the second case, two NX5000 modules also form a NIC Teaming redundant pair, interconnected in two different switches (Ethernet HSDN A and Ethernet HSDN B). In some point, these two switches must be interconnected, for the two NIC Teaming ports connection and for an even higher availability (against double failures).

Such Ethernet architectures turn possible an excellent availability, against Ethernet port failures, in cables and switches.

ATTENTION

If two modules, or Ethernet interfaces, form a NIC Teaming redundant pair, the configuration and device inclusion will be only possible in the first interface. The second interface will have his configuration parameters blocked for edition.

A cluster of two Ethernet ports forming a NIC Teaming pair has a single IP address, related to the port pair. This way, a client as SCADA or MasterTool, connected to a CPU server, doesn't need to worry in IP address changing in case there's a failure in any NIC Teaming pair port.

Each of the Ethernet interfaces that form the NIC Teaming pair have an unique diagnostics structure to point to failures which eventually might appear in any port of a NIC Teaming pair.

For further details regarding NIC Teaming configuration and diagnostics, see the following sections:

- Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2)
- NX5000 Modules Configuration

6.3.11. IP Change Methods

A redundant cluster from Nexto Series has four methods for IP change in the Ethernet ports of the NX5000 modules in each half-cluster and one method for IP change in the NET 1 and NET 2 ports of the NX3030 CPU. These methods define the ports' behavior, regarding its IP, according to the current state of the half-cluster (Active or Non-Active) and with the half-cluster (PLCA or PLCB).

The methods are: Fixed IP, Exchange IP, Active IP and Multiple IP.

Overall, it can be listed up to four IPs, according to the IP change method.

6.3.11.1. Fixed IP

It's the simplest method for IP addressing and can be configured in the Ethernet interfaces in the NX5000 Ethernet modules. In this method, it's only listed the IP addresses from the PLCA and from PLCB. Apart from the redundancy state, PLC Active or Non-Active, the PLCA will always answer by the configured IP, as also will PLCB.

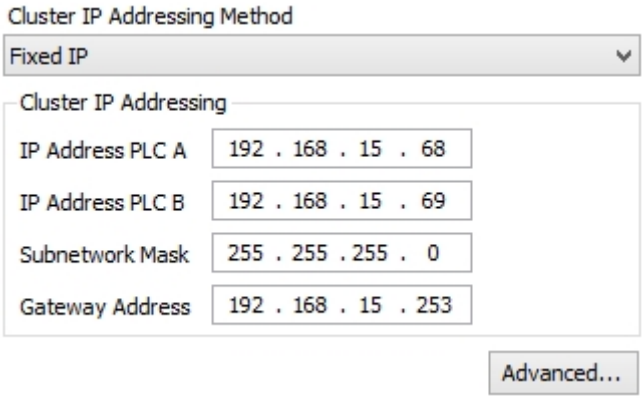


Figure 167: Fixed IP method

Parameters that must be configured in the Fixed IP method:

- IP Address PLC A: PLCA communication address
- IP Address PLC B: PLCB communication address
- Subnetwork Mask
- Gateway Address

6.3.11.2. Exchange IP

The Exchange IP can be configured in the Ethernet interfaces in the NX5000 Ethernet module. In this method, the half-cluster IP depends on the PLC state (Active or Non-Active). On every switchover the IP change occurs between the half-clusters allowing them to assume the IP address from the new redundancy state.

PS: for this addressing method, the Ethernet ports from both PLCs (PLCA and PLCB) assume the same IP address while they both are in the Non-Active state, generating a network address conflict. Considering this situation uncommon, where no PLC is controlling the system, this turns out to be a big problem and has to be considered.

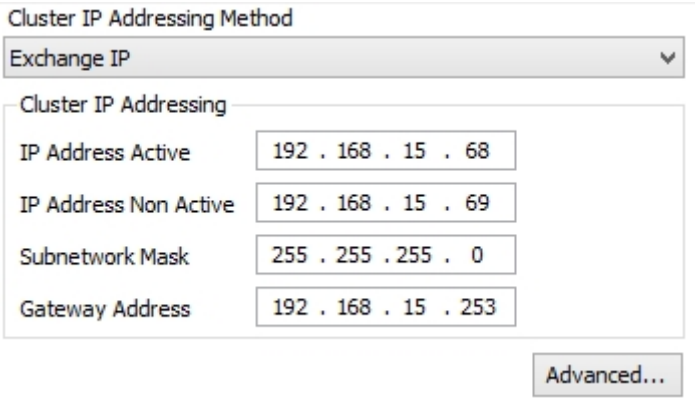


Figure 168: IP Automatic Change

Parameters that must be configured in the Exchange IP method:

- IP Address Active: PLCA communication address
- IP Address Non Active: PLCB communication address
- Subnetwork Mask
- Gateway Address

6.3.11.3. Active IP

This method is used in the redundant NX3030 CPU NETs and it's also possible to be configured in the NX5000 modules. In this method there's an IP for the Active half-cluster and two more IPs, one for the PLCA and another for the PLCB. In the redundant NX3030 CPU NETs, the Active IP address is added to the interface of the Active PLC, and it can use either the Active IP address or the PLCX IP address in order to establish communication with the PLC. On the other hand, in the NX5000 Ethernet modules the Active IP address substitutes the Non-Active PLCX IP address, when the PLC is in Active mode.

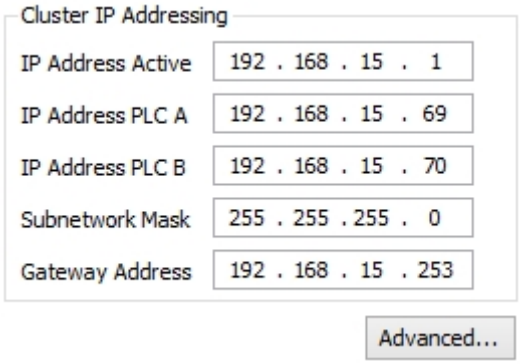


Figure 169: Active IP method – Redundant NX3030

Parameters that must be configured in the Active IP method for the NETs of a redundant NX3030 CPU:

- IP Address Active: IP address added to the interface when the PLC is in Active state
- IP Address PLC A: PLCA communication address, apart from its current state
- IP Address PLC B: PLCB communication address, apart from its current state
- Subnetwork Mask
- Gateway Address

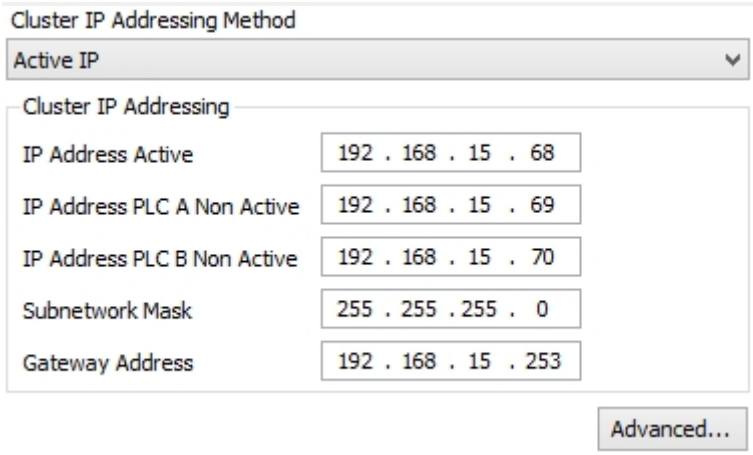


Figure 170: Active IP method – NX5000

Parameters that must be configured in the Active IP method for the NX5000 Ethernet modules:

- IP Address Active: Active PLC communication address. Replaces the IP address from the Non-Active PLCX
- IP Address PLC A Non Active: PLCA communication address, when in Non-Active state
- IP Address PLC B Non Active: PLCB communication address, when in Non-Active state
- Subnetwork Mask
- Gateway Address

6.3.11.4. Multiple IP

The Multiple IP method can be configured in the Ethernet interfaces from the NX5000 Ethernet modules. In this method there's an IP for each half-cluster and for each state of the PLC. The PLCA assumes an IP address when it's Active and another when it's Non-Active. The same happens for the PLCB regarding its state (Active or Non-Active).

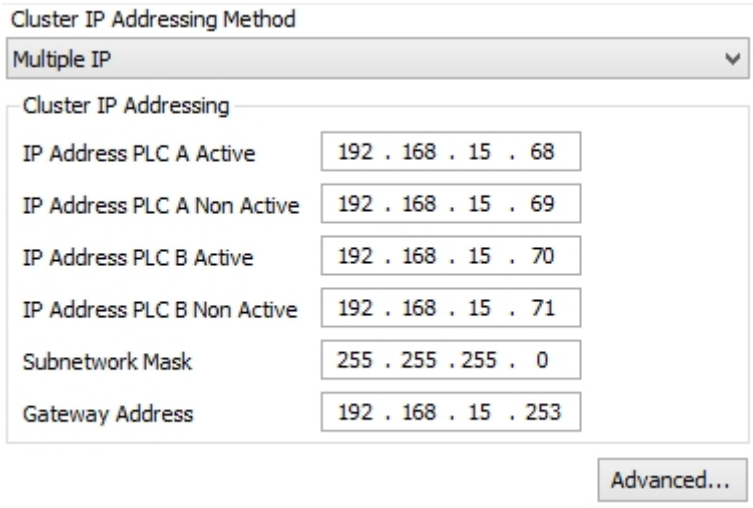


Figure 171: Multiple IPs method

Parameters that must be configured in the Multiple IP method:

- IP Address PLC A Active: PLCA communication address, when in Active state
- IP Address PLC A Non Active: PLCA communication address, when in Non-Active state
- IP Address PLC B Active: PLCB communication address, when in Active state
- IP Address PLC B Non Active: PLCB communication address, when in Non-Active state
- Subnetwork Mask
- Gateway Address

6.3.12. NIC Teaming and Active IP Combined Use

In case a determined port pair form a NIC Teaming in a redundant CPU, these ports can implement, at the same time, the strategies NIC Teaming and Active IP.

E.g. if the NX3030 CPU NET 1 and NET 2 ports form a NIC Teaming pair, then:

- IP Address PLC A: IP address of the NET 1 + NET 2 ports in the PLCA NX3030 CPU
- IP Address PLC B: IP address of the NET 1 + NET 2 ports in the PLCB NX3030 CPU
- IP Address Active: IP address of the NET 1 + NET 2 ports in the NX3030 CPU in the Active CPU

This way, the excellent availability from the NIC Teaming strategy is associated with the practicality of the Active IP strategy, which doesn't need scripts in SCADA systems or in other clients connected to the Active CPU server.

6.3.13. Ethernet Interfaces Use with Vital Fault Indication

The Ethernet ports of NX3030 and NX5000 modules can be configured to generate vital failures. This option is important for applications in which the modules of inputs and outputs are distributed over Ethernet network. In this case, if a failure occurs on the Ethernet port, this will generate a switchover. This behavior is applicable only to Ethernet ports where there is at least a communication driver that manages fault.

The communication drivers that generate vital failure are MODBUS Client and MODBUS Symbol Client (all references to MODBUS Client in the following sections apply to both cases). The MODBUS Server drivers, MODBUS Symbol Server and EtherCAT Master do not generate vital failure. Thus, if an Ethernet port has a MODBUS Client driver configured and a failure occurs in the Ethernet port, a switchover will be generated if vital fault option is enabled. If the driver configured on the Ethernet port is a MODBUS Server, even if there is failure in the door, it will not generate a vital failure that causes a switchover.

To a fault be considered a vital failure in an Ethernet port on a MODBUS Client, all servers configured in the driver must be faulty. That is, if there is more of a MODBUS Client driver configured in the same Ethernet port, is considered vital failure when all servers of both Clients are faulty.

When the Ethernet port is configured to operate with NIC Teaming, the vital failure will be considered only when the two pair of doors fail.

6.3.13.1. Failure in Ethernet Interface

A switchover can be generated due to failure in the Ethernet interface, such as a loss of link. The link loss may be caused, for example, by a cable breakage or failure of a switch on the Ethernet network. Accordingly, it is necessary that, in addition to being configured to generate vital failure, there is a MODBUS Client instance configured on the Ethernet interface.

When the interval of MainTask is greater than or equal to 100 ms after the fault is detected the switchover will occur in up to two cycles of MainTask. When the interval of MainTask is less than 100 ms switchover will occur within 100 ms plus the time of MainTask after detection of failure.

6.3.13.2. Failure in Connected MODBUS Server

The time to detect the fault in a remote MODBUS Server depends on the time-out settings configured on each MODBUS Client. When a fault is detected in all Servers, the bAllDevicesCommFailure diagnostic (see Modbus Diagnostics used at Redundancy) section used in) changes its state to TRUE. When this happens, the switchover will happen 3 seconds after this transition.

6.3.14. OPC DA Communication Use with Redundant Projects

The OPC DA protocol can be configured to communicate with redundant clusters over SCADA systems. When this option is selected in the creation of a redundant project, the *Symbol Configuration* object is added to the project. In this object are set system variables that will be sent to the SCADA system. This communication option is enabled in the CPU of the Ethernet ports NX3030. For further information related to the configuration of an OPC communication with redundant projects, refer to the Configuration with the PLC on the OPC DA Server with Connection Redundancy section of this Manual.

6.3.15. Redundant CPU States

In a redundant system, a CPU (PLCA or PLCB) may assume the following states:

- Active
- Stand-by
- Inactive
- Not-Configured
- Starting

> **ATTENTION**
>
> Frequently this manual will use the designation "Non-Active" for each state different from Active, in other words, to design any one from the other 4 states (Stand-by, Inactive, Not-Configured and Starting). An Active CPU is the one that is in Active state and a Non-Active CPU is the one that isn't in Active state.

In the following sections these five states are briefly described. Further details regarding the redundant CPU states are described in the *Transition between Redundancy States* section, when the state machine and the transition causes are also described.

6.3.15.1. Not-Configured State

This is the initial redundancy state. The CPU is found in this redundancy state:

- By convention, while the CPU is OFF
- Before starting the MainTask
- Before the Starting state is switched
- In case there's a restart through a command as Reset Warm, Reset Cold or Reset Origin

In case the MainTask is being executed in the Not-Configured state, the following tasks are executed:

- The PROFIBUS masters are disabled
- The cyclic synchronization services are executed (see Cyclic Synchronization Services through NETA and NETB section), if the conditions for its execution are true
- The sporadic synchronization services can also be executed (see Sporadic Synchronization Services through NETA and NETB section)

The CPU will be blocked in the Not-Configured state if the other CPU is in Active state, and this CPU project is different from the Active CPU project (except if the project automatic synchronization is disabled – see Project Synchronization Disabling). In case this situation doesn't occur, a transition from the Not-Configured state to the Starting state happens as soon as a configuration request arrives.

Sometimes, the CPU goes to Not-Configured state when has already received an automatic configuration request, when the new request for Starting state changing is not necessary. This happens at the CPU energizing, for instance.

In other situations, the user must request manually this configuration, e.g. pressing a button on the PX2612 redundancy command panel. Manually configuration requests typically aren't necessary when an user maintenance is needed before going out from the Not-Configured state, e.g. if the CPU hasn't reached the Not-Configured state due to some failure.

After getting out from the Not-Configured state, the PLC can go back to this state, due to events such as:

- Restarting (Reset Warm, Cold or Origin)
- PLC switch off
- Different projects between this PLC and the Active PLC

6.3.15.2. Starting State

Different from all other 4 states which can last indefinitely, the Starting state is temporary, taking only a few seconds. This state is always reached from the Not-Configured state, through a configuration request.

At the beginning of the Starting state, several actions, tests and verifications are executed, in order to decide which will be the next state:

- PROFIBUS masters are enabled in a passive state. The passive mode is used to test the transmission and reception PROFIBUS circuits and the physical layer, to avoid an occult failure to happen
- Verify if the CPU identification is correct (must be PLCA or PLCB)
- Verify if there are problems in the configuration parameters extracted from MasterTool project
- Verify the NX4010 module integrity
- The cyclic synchronization services are executed (see Cyclic Synchronization Services through NETA and NETB section), if the conditions for its execution are true
- Verify the firmware compatibility version between both CPUs
- Verify if the projects from both CPUs are equal, if the project automatic synchronization is enabled (see Project Synchronization Disabling section)
- In case the other CPU is in Active state, verify the possibility to establish a passive PROFIBUS communication with it. The passive mode is used to test the transmission and reception PROFIBUS circuits and the physical layer, to avoid an occult failure to happen
- In case the other CPU is in unknown state due to failures in NETA and NETB, verify the possibility of establishing a passive PROFIBUS communication with it. If there is no PROFIBUS network in the project and it neither has the PX2612 Panel, check if CPU's NET1/NET2 are receiving Keep Alive packages from the other half-cluster.

Depending on the results of these verifications and tests, the CPU can go from the Starting state to any from the other four states.

6.3.15.3. Active State

In this state, the CPU controls the automated process, using the ActivePrg program, executed only in this state. The Active CPU also updates the PROFIBUS remote I/O system, putting its PROFIBUS masters in active state. The active state is used to establish communication with the PROFIBUS remotes (slaves).

The Active CPU also verifies its internal diagnostic and user switchover requests to determine if a switchover is necessary. The CPU goes out from the Active state only if it knows the other CPU is in Stand-by mode, and able to assume as Active.

However, there are some situations where the Active CPU could go out from the Active state even with no certainty that the other CPU is in Stand-by state (e.g. if the CPU is switched off).

6.3.15.4. Stand-By State

In this state the CPU is ready to be switched to the Active state, in case there's a request for that, as a failure in the Active CPU.

The Stand-by CPU also verifies its own diagnostics and can be switched to the Not-Configured or Inactive state, in case some failures occur.

PROFIBUS masters are enabled in the passive state. The passive mode is used to test the transmission and reception PROFIBUS circuits and the physical layer, to avoid an occult failure to happen. Total failure in PROFIBUS networks configured as vitals cause a switching to the Inactive state. A total failure in a PROFIBUS network damages both composing networks (redundant PROFIBUS network) and the single composing network (non-redundant PROFIBUS network).

If the Ethernet interfaces are enabled with vital failure option, clients are enabled in passive state. Total failures in Ethernet networks configured as vital cause a switch to the Inactive status. A total failure in an Ethernet network reaches the two networks that comprise (Redundancy of Communication option enabled) or the only network that compose (Redundancy of Communication option disabled).

6.3.15.5. Inactive State

This state is normally reached after some failure types, or due to a manual request before a programmed maintenance.

PROFIBUS masters are enabled in the passive state. The passive mode is used to test the transmission and reception PROFIBUS circuits and the physical layer, to avoid an occult failure to happen.

Before switching to another state, first the diagnosed failures must be corrected or the programmed maintenance executed, if those have driven the CPU to Inactive state. After, a transition for the Not-Configured state must be done, requesting a configuration. Then, a switch to the Starting state must be executed. After the Starting state, the CPU can:

- Return to the Inactive state, if determine failure types remain
- Return to the Not-Configured state, in case of other failure types
- Go to Stand-by state, if the other CPU is in Active state
- Go to Active state, if the other CPU isn't in Active state

6.3.16. PX2612 Redundancy Command Panel Functions

The PX2612 redundancy command panel is shown on Figure 162, while Figure 163 shows its frontal view with more details. Besides this, Figure 164 shows how this panel must be connected to the PLCA and PLCB half-clusters.

The PX2612 is divided in two sections: one controlled by PLCA and another by PLCB. These controllers are possible through cables AL-2317/A for PLCA and AL-2317/B for PLCB, and allow each CPU to read three buttons, write on three LEDs and a NO relay contact.

Observing the frontal view on Figure 163:

- PLCA executes the STAND-BY and INACTIVE buttons reading in PLC A sector
- PLCA executes the TURN ON PLC B button reading
- PLCA executes the writing on the three LEDs (ACTIVE, STAND-BY and INACTIVE) from the PLC A sector
- PLCA executes the writing on the RL B relay, used to switch off PLCB
- PLCB executes the STAND-BY and INACTIVE buttons reading in the PLC B sector
- PLCB executes the TURN ON PLC A button reading
- PLCB executes the writing on the three LEDs (ACTIVE, STAND-BY and INACTIVE) from the PLC B sector
- PLCB executes the writing on the RL A relay, used to switch off PLCA

6.3.16.1. PX2612 Buttons

This section describes the functions of the PX2612 buttons.

The STAND-BY button has the following functions:

- To request a switching from the Active state to the Stand-by state, useful when maintenance in the Active CPU is needed. After the Active CPU is switched to Stand-by (and consequently the Stand-by CPU is switched to Active), it's possible to switched from Stand-by to Inactive using the INACTIVE button, and then execute the programmed maintenance in the inactive state
- To request a configuration which causes a switching from the Not-Configured to the Starting state, typically after the failures that caused the transition to the Not-Configured state are repaired. After the Starting state, normally the CPU is supposed to go to the Stand-by state (or Active, if the other CPU isn't in the Active state)
- To request a switching from the Inactive state to the Not-Configured state requesting a configuration already. This occurs typically after the failures which caused the transition to the Inactive state were corrected. After the Not-Configured state, the configuration must take it to the Starting state. After the Starting state, normally the CPU is supposed to go to the Stand-by state (or Active, if the other CPU isn't in the Active state)

The INACTIVE button requests a switching from the Stand-by state to the Inactive state, which can be useful to execute a programmed maintenance in the Stand-by CPU. After this maintenance, the STAND-BY button may be used to make it go back to the Stand-by state, passing by the Not-Configured and Starting state (see previous description of the STAND-BY button).

The TURN ON PLCx (x = B for PLCA, or x = A for PLCB) button is used to cause a reactivating in the other CPU, in case the local CPU has switched off. As it is described in the *Transition between Redundancy States* section.

There are exceptional situations when a CPU switches off the other at assuming the Active state, in order to avoid the possibility of both CPUs to assume the Active state simultaneously.

ATTENTION

For a button to be considered, it must be pressed for at least 1 second. Furthermore, during this second, only this button must be pressed (the other 2 buttons must be released).

ATTENTION

There are alternative ways to generate the same effects of the STAND-BY, INACTIVE and TURN ON PLCx buttons. Commands generated by the local CPU or the remote CPU can be used, as described, preliminary, in the *Diagnostics, Commands and User Data Structure* section. A more detailed description of these commands can be found in the Redundancy Commands section.

6.3.16.2. PX2612 LEDs

The PX2612 LEDs are used to inform the redundancy state, as shown on the following table below:

Redundancy state	LED ACTIVE	LED STAND-BY	LED INACTIVE
Not-Configured	off	off	off
Starting	on	on	on
Active	on	off	off
Active (recent)	blinking	off	off
Active (switching off the other CPU)	on	blinking	off
Active (recent and switching off the other CPU)	blinking	blinking	off
Stand-by	off	on	off
Inactive	off	off	on

Table 202: PX2612 LEDs

Each LED can be off, on or blinking. In case it's blinking, it remains on for 0.5 seconds and off for the same time.

Note that there are four different animations for the Active state, due to the following features:

- At the first 2 seconds in Active state the ACTIVE LED blinks and remains on afterwards. This animation was created because in the first instants of the Active state, the CPU won't accept commands to get out from this state. For further details regarding this Active CPU behavior, see *Transition between Redundancy States* and First Instants in Active State sections
- In case this CPU is switching off the other CPU through its PX2612 relay, the LED STAND-BY blinks. It remains off otherwise

6.3.16.3. PX2612 Relays

The PX2612 has two NO relays. The PLCA can control the RL B, to command the PLCB switching off. The PLCB can control the RL A, to command the PLCA switching off.

Such switching off situations happen in exceptional situations, described in the *Transition between Redundancy States* section.

6.3.17. Transition between Redundancy States

The following figure shows the redundancy state machine, illustrating all the possible transitions between redundancy states.

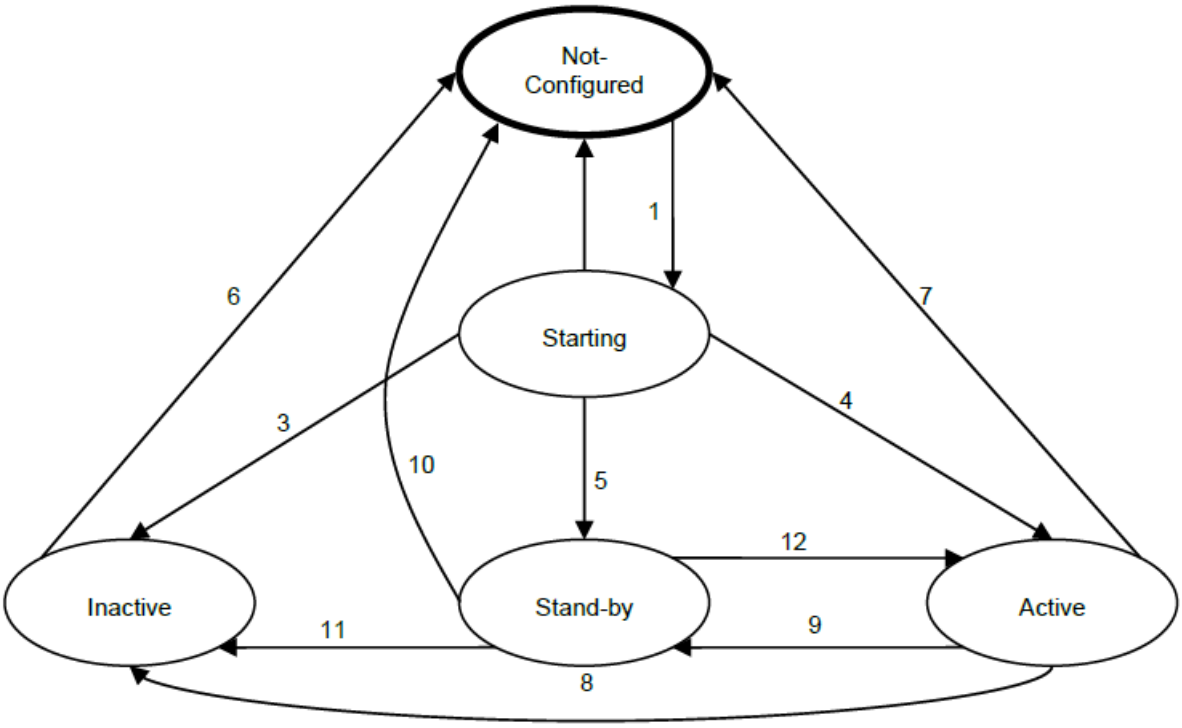


Figure 172: Redundancy State Machine

The following sub-sections describe all these transitions, and the causes which can trigger them. In order to interpret correctly this state machine functioning, some rules and sequences must be established:

- Transitions which originate from the same state must be analyzed in the sequence established by their number. E.g. the transitions 2, 3, 4 and 5 are originated from the Starting state. In this example, the transition 2 is first analyzed, then 3, 4 and, finally, 5. In case the transition 2 is triggered, the transitions 3, 4 and 5 won't be analyzed
- Inside a specific sub-section describing a transition, several conditions can trigger it. These conditions must be analyzed in the sequence they appear in the sub-section. Any condition that goes true can cause a transition. If a condition causes a transition, the next conditions don't need to be analyzed
- Transitions can only be triggered if the CPU is on and the MainTask is executing. Otherwise the CPU is assumed to be in the Not-Configured state
- In several cases, transitions caused by the PX2612 panel buttons are mentioned. It must be recalled there are alternatives for these buttons, which are internal commands from one CPU or the other (via NETA / NETB). Such commands were mentioned preliminary in the *Diagnostics, Commands and User Data Structure* section and are better described in the Redundancy Commands section. In the following sub-sections, to simplify, these commands are not mentioned, but one must remember they can cause the same transitions as the PX2612 button

6.3.17.1. Transition 1 – Not-Configured to Starting

ATTENTION

The conditions of this sub-section must not be analyzed in case the other CPU is in Active state and the projects are different. This CPU must remain in the Not-Configured state while its project is different from the other CPU project, if the other is in Active state. This note isn't valid if the project automatic synchronization is disabled (see Project Synchronization Disabling section), as in this case differences between the CPUs projects are allowed.

- A configuration request is already existent at the beginning of the Not-Configured state. This occurs in the moment the CPU is switched on, and also other situations, described in the next sub-sections
- The STAND-BY button was pressed during the Not-Configured state. This causes a manual configuration request. The user typically presses STAND-BY after failure repairing which had driven this CPU to the Not-Configured state

6.3.17.2. Transition 2 – Starting to Not-Configured

- This CPU was turned off or restarted (Reset Warm, Cold or Origin) or its CPU went to Stop mode
- The identification register of this CPU is invalid (different than PLCA or PLCB)
- There are logic configuration errors in the project received from MasterTool IEC XE
- The other CPU is in the Active state and the firmware version in this CPU is incompatible with firmware version in it
- The other CPU is in Active state and the project in this CPU is different from the project in it. Besides going to the Not-Configured state, a configuration request is made. This way, after the projects are synchronized, the CPU goes out automatically from the Not-Configured state to the Starting state. This condition isn't analyzed if the project automatic synchronization is disabled (Project Synchronization Disabling section)

6.3.17.3. Transition 3 – Starting to Inactive

- NX4010 module not detected in the bus, or its microprocessor failure
- A synchronism channel (NETA or NETB) is in failure and this CPU knows this failure was caused by hardware components or internal software (internal failures of NETA or NETB)
- The other CPU is in Active state. However, it's not possible to synchronize the redundant data or the redundant forcing list
- The other CPU state cannot be discovered through NETA / NETB, but this CPU can monitor the traffic in some configured PROFIBUS networks in vital failure mode. This way, it looks like the other CPU is controlling the process, even though NETA / NETB aren't working to confirm it
- On the redundancy without PX2612 panel and without PROFIBUS network, in case other PLC's state can't be known via NETA/NETB, but this PLC is receiving information that the other cluster's state is ACTIVE through Keep Alive packages received via NX3030's NET1 or NET2
- Link loss occurred to an Ethernet Interface configured as Vital Failure.

6.3.17.4. Transition 4 – Starting to Active

- The other CPU is in Non-Active state. Before the transition is possible, this condition must remain true for some time, higher to PLCB than PLCA. This way, at the moment PLCA and PLCB are simultaneously turned on; PLCA has priority to take over in Active state
- The other CPU state can't be discovered through NETA / NETB, and besides that this CPU can't monitor traffic in any PROFIBUS network configured as vital failure mode, or those networks weren't created. Therefore, it really looks the other CPU if off or out of execution. For safety reasons, besides switching to Active, this CPU turns the other off using its PX2612 relay. This condition must be kept for a while before the transition is executed

6.3.17.5. Transition 5 – Starting to Stand-by

- The other PLC is in Active state. The redundant data synchronization and the redundant forcing list synchronization services are working correctly

6.3.17.6. Transition 6 – Inactive to Not-Configured

- This PLC was switched off or restarted (Reset Warm, Reset Cold or Reset Origin) or its CPU went to Stop mode
- The STAND-BY button was pressed on the PX2612. Besides going to the Not-Configured state, a configuration request is made. This way, the CPU goes out automatically from the Not-Configured state for the Starting state. The user typically presses this button after repairing the failure which has driven the CPU to the Inactive state
- This PLC has its synchronization disabled and the project is different from the Active PLC, at the STAND-BY button pressing, the PLC goes from Inactive to Not-Configured

6.3.17.7. Transition 7 – Active to Not-Configured

- This PLC was switched off or restarted (Reset Warm, Reset Cold or Reset Origin) or its CPU went to Stop mode

6.3.17.8. Transition 8 – Active to Inactive

- NX4010 module not detected in the bus, or its microprocessor failure. This CPU knows the other CPU was in Stand-by state before this failure happened. This condition isn't analyzed in the first 2 seconds in Active state
- This PLC has lost communication with another PLC through NETA and NETB due to an internal failure but knows the other PLC was in Stand-by mode just before the failure occurred. This condition isn't analyzed in the first 2 seconds in Active state
- This CPU can't control all PROFIBUS networks configured in vital failure mode and knows the other CPU is in Stand-by state. This condition isn't analyzed in the first 2 seconds in Active state
- This CPU detected a total failure in Ethernet networks configured in vital failure mode, and knows that the other CPU is in Stand-by state

6.3.17.9. Transition 9 – Active to Stand-by

- Both PLCs, for some reason, are in Active state and this conflict must be solved. The PLCA switches to Stand-by state in case this conflict remains. The PLCB does the same after a delay smaller than PLCA. This way, in this case, PLCA has priority to remain in Active state
- The STAND-BY button was pressed and this CPU knows the other CPU is in Stand-by state. This condition isn't analyzed in the first 2 seconds in Active state

6.3.17.10. Transition 10 – Stand-by to Not-Configured

- This PLC was switched off or restarted (Reset Warm, Reset Cold or Reset Origin)
- The other PLC is in Active state and it's known this PLC project is different from the Active PLC. Besides going to the Not-Configured state, a configuration request is made. This way, after the projects synchronization, the PLC goes automatically from the Not-Configured state to the Starting state. This condition isn't analyzed if the project automatic synchronization is disabled (Project Synchronization Disabling section)
- The other PLC is in Active state and firmware version of this PLC is incompatible with the firmware version of the Active PLC

6.3.17.11. Transition 11 – Stand-by to Inactive

- NX4010 module not detected in the bus, or its microprocessor failure
- The INACTIVE button was pressed on the PX2612. This is made typically in order to execute a programmed maintenance in the Non-Active CPU. Any programmed maintenance must be avoided in the Stand-by CPU, thus is recommended to switch to Inactive mode
- The other CPU is in Active state. However the redundant data synchronization or the redundant forcing list synchronization services haven't worked in last four cycles of the MainTask or the diagnostics synchronization service haven't worked in the last two cycles of the MainTask
- The other PLC is in Active state. However, this PLC can't monitor traffic in every PROFIBUS network configured as vital failure mode
- The other CPU is in Active state. However, this CPU detected failure in Ethernet ports configured as vital failure mode

6.3.17.12. Transition 12 – Stand-by to Active

- The other CPU state is unknown due to NETA and NETB failures. In this case, besides going to Active state, for safety reasons, this CPU switches off the other CPU using the PX2612 relay. When the Redundancy does not use PX2612 panel and there isn't PROFIBUS DP, the CPU uses a Keep Alive mechanism through NX3030's NET1/NET2 ports, to intercommunicate the state between PLCs and detect that the ACTIVE one isn't controlling the process anymore.
- The other CPU state is known and different than Active

6.3.18. First Instants in Active State

In the first 2 seconds in Active state, as already described in PX2612 Redundancy Command Panel Functions section, the LED ACTIVE blinks and remains on after this time has passed.

While the LED ACTIVE blinks, several transitions which, usually, could take the CPU from the Active state, aren't analyzed (see previous sub-sections that define transitions from the Active state). E.g. during this time, it doesn't work to press the STAND-BY button to try and make the CPU go to Stand-by state.

Only two conditions allow the CPU to go out of the Active state while the LED ACTIVE blinks. They are the following:

- This PLC was switched off or restarted (Reset Warm, Reset Cold or Reset Origin), causing a transition to Not-Configured state
- Both PLCs, for some reason, are in Active state and this conflict must be solved. The PLCA switches to Stand-by state in case this conflict remains. The PLCB does the same after a delay smaller than PLCA. This way, in this case, PLCA has priority to remain in Active state

Furthermore, in the very first instants that a PLC assumes the Active state, some non-redundant diagnostics may not be valid, such the diagnostics of the NX5000 and NX5001 modules. The method used to ignore the diagnostics possibly invalid is described in section Reading Non-Redundant Diagnostics.

6.3.19. Common Failures which Cause Automatic Switchovers between Half-Clusters

In this section, the more common failures which, automatically, cause a switchover from the Active CPU to Non-Active and from Stand-by CPU to Active CPU are listed. These failures trigger a sub-group of those transitions examined in the *Transition between Redundancy States* section.

- Power supply fault in the Active CPU. It's important that both CPUs have redundant power supplies, in order to avoid that a power supply failure doesn't affect the Stand-by CPU
- NX8000 power supply fault in the Active CPU
- Rack bus failure (NX9000, NX9001, NX9002 or NX9003) in the Active CPU
- Failures in the NX3030 CPU from the Active CPU, such as:
  - Watchdog
  - Restart (Reset Warm, Cold or Origin)
  - Stop
  - Failure in the bus interfaces in one or both synchronization channels NETA and NETB
- Failures in the NX4010 from the Active PLC, such as:
  - Not recognized module in the NX3030 CPU bus
  - Failure in the NX4010 microprocessor which prevents the NETA/NETB and the PX2612 control panel (buttons, LEDs and relay) internal diagnostics updating
  - Internal failures that affect one or both synchronization channels NETA and NETB
- Active PLC PROFIBUS network total failure, in case this network is configured in vital mode. In case the PROFIBUS network is redundant, both composing networks must fail (double failure)
- Total failure of an Ethernet network in active CPU, if this network is configured with vital failure. If the Ethernet network is redundant, both networks that compose it must be faulty (double failure)

6.3.20. Failures Associated to Switchovers between Half-Clusters Managed by the User

Among the described transition in the Transition between Redundancy States section, some turn possible the user to manage switchovers between half-clusters, due to failures that don't generate automatic switchovers.

There are very particularly cases which depend on the philosophy of each client. E.g.: a case where the SCADA system loses the communication with the Active CPU, but keeps communicating with the Stand-by CPU.

Some clients would rather to have a manual switchover, where the operator presses the PX2612 STAND-BY button, to the Active CPU. The switchover causes a communication retry with the new Active CPU.

An alternative solution would be to cause a switchover by sending a command from the SCADA system to the Stand-by CPU, which would transmit to the Active CPU through NETA/NETB, using the RedCmdLocal (Stand-by CPU) and RedCmdRem (Active CPU) data structures to transport a command equivalent to the PX2612 STAND-BY button.

It would be also possible the Active CPU detect its communication lost with the SCADA system itself and to activate a command in the RedCmdLocal, equivalent to the PX2612 STAND-BY button. This would be a totally automatic solution with no operator intervention that would be typically made in the ActivePrg POU.

Through data structures described in the Diagnostics, Commands and User Data Structure section, it's possible to exchange diagnostics and commands between the half-clusters through NETA and NETB. This way, the user can execute special redundancy managing for failures that normally wouldn't cause any switchover. Further details regarding these data structures are offered in the following sections:

- Redundancy Diagnostics Structure
- Redundancy Commands
- User Information Exchanged between PLCA and PLCB

Below, is exemplified how the user can manage failures and execute a switchover due to an error in the Ethernet interfaces from the Active PLC (this code should be used in the ActivePrg POU):

```
//Verify if NIC Teaming is enabled.
IF ((DG_NX3030.tDetailed.Ethernet.NET1.szIP = '0.0.0.0') OR (DG_NX3030.tDetailed
    .Ethernet.NET2.szIP = '0.0.0.0')) THEN
//NIC Teaming enabled: error in two NETs to execute a switchover.
IF (DG_NX3030.tDetailed.Ethernet.NET1.bLinkDown AND DG_NX3030.tDetailed.Ethernet
    .NET2.bLinkDown) THEN
//Change the local PLC to StandBy..
DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal := TRUE;
END_IF
ELSE
//NIC Teaming disabled: error in one of NETs to execute a switchover.
IF (DG_NX3030.tDetailed.Ethernet.NET1.bLinkDown OR DG_NX3030.tDetailed.Ethernet.
    NET2.bLinkDown) THEN
//Change the local PLC to StandBy.
DG_NX4010.tRedundancy.RedCmdLoc.bStandbyLocal := TRUE;
END_IF
END_IF
```

ATTENTION

When two Ethernet interfaces form a NIC Teaming pair, the inactive interface will always have the IP address 0.0.0.0. This isn't a valid IP and is no possible to configure manually an interface with this address.

6.3.21. Fault Tolerance

The main objective of a redundant CPU is the system availability increase. The availability is the ratio between the time while the system is working properly and the total time since the system has been implemented. For instance, if a system was implemented 10 years ago and during this time, wasn't working due to failures for a year, then its availability was only 90%. This kind of availability is usually unacceptable for critic systems, where 99.99% availability is required, or even more.

In order to reach this availability level, several strategies are necessary:

- Utilization of more reliable components (with high MTBF or Mean Time between Failures), contributing for the MTBF increase of the system as a whole
- Utilization of redundancy for, at least, the most critical components or components with smaller MTBF, in such a way that a component failure can be tolerated without stopping the system. If the redundancy is implemented through components duplication, it will be necessary that both fail for the system as a whole become unavailable
- High diagnostics coverage, especially in redundant components. The component redundancy isn't very useful for the availability increase when is not possible to discover which component failed. In this case, the first failure in one component still doesn't drop the system, but remains hidden, until the second failure occurs, dropping the system, as the first failure wasn't yet repaired. The failures can be classified between diagnosable and hidden. It's strongly recommended that all redundant components failures are diagnosable
- It's also important that non-redundant components have wide diagnostics coverage, as, frequently, the system can continue working even with a non-redundant component failure. The component may not being requested, e.g. a relay with NO contact which rarely has its coil activated, doesn't have its failure detected until the moment the system requires its closing
- Low repair time for non-redundant components. A non-redundant component failure can drop the system, and during the repair, the system will be unavailable
- Possibility of repairing or substituting a redundant component without stopping the system. If this possibility exists, a great availability increase it got. Otherwise, a stop must be programmed in order to substitute the component and the repair time is computed as unavailable time
- Low repair time for redundant components. A redundant component failure doesn't drop the system, but during its repair, a failure in its redundant pair could happen. For this reason, it's important that the failure is repaired quickly after diagnosed. The higher the repair time, the higher the probability of a second failure to occur in the redundant component during this time, what would drop the system. Therefore, the higher the repair time, the lower the system availability

- Program periodic offline tests in components in order to detect not automatically diagnosable failures by the system. The objective is to detect hidden failures, especially in redundant components or simple components which aren't being requested (e.g. a security relay). Offline tests, sometimes, imply in system stopping what decreases the availability. Normally, special situations, such as process programmed maintenance, are used for that purpose. The higher the period between offline tests, the higher the time which the failure may remain hidden, and the higher the probability of a failure to damage the system, in other words, the smaller the availability

These principles were considered in the redundant CPU project using NX3030.

The next sub-sections analyze several failure types and how they are tolerated or not, and if there are switchovers associated to the tolerated failures.

6.3.21.1. Simple Failure with Unavailability

Some components, as they aren't doubled, don't even tolerate a simple failure without causing some kind of unavailability. In a redundant CPU using CPU NX3030, this is related to the following components:

- PROFIBUS remotes (slaves) in a non-redundant PROFIBUS network
- Ethernet remotes (slaves) in a non-redundant network
- I/O Modules

The failure intolerance of a non-redundant PROFIBUS network can be solved if a redundant PROFIBUS network is used, which is advisable in systems that demand a high failure tolerance. Figure 159 shows an example of a redundant PROFIBUS network architecture. Likewise intolerance to failure of a non-redundant Ethernet network can be solved by using a redundant Ethernet network configuration with NIC Teaming.

Regarding the I/O module unavailability, it must be observed that it doesn't imply total system unavailability. It constitutes a partial unavailability, only in the control mesh that uses this I/O module.

Even though there's no redundancy prevision for I/O modules, the user application can manage it in special cases. E.g. the user can insert 3 analog input modules in 3 different PROFIBUS remotes, and implement a vote scheme between analog inputs triples, for a critic system. However, as mentioned, such solutions must be managed by the user. There's no automatic support for them. Such solutions, generally speaking, also imply in the field transducers and actuators redundancy.

6.3.21.2. Simple Failure without Unavailability Causing a Switchover

Some redundant components tolerate simple failures without causing unavailability, but cause switchover:

- Racks (NX9000, NX9001, NX9002 or NX9003)
- Power Supply (NX8000)
- CPUs (NX3030)
- NX4010 modules
- NX5001 modules (PROFIBUS masters) in non-redundant PROFIBUS network configuration
- NX5000 modules (Ethernet) in configurations without NIC Teaming
- PROFIBUS slave interface in a redundant remote (PO5063V5, PO5065, NX5210 or AL-3416). In this case, different from the previous, the switchover happens inside the remote, between the PROFIBUS A and B networks.

ATTENTION

In case of failure of the CPU NX3030 or NX4010 module in architectures where panel PX2612 or PROFIBUS network is not used, the CPU will remain in its current state. In this case, if the failure occurs in the half-cluster active, system downtime occurs.

6.3.21.3. Double Failure without Unavailability Causing a Switchover

Some components are doubled in each half-cluster, this way, before causing a switchover, both must fail:

- NX5001 modules (PROFIBUS masters) in redundant configuration, configured in vital failure mode.
- NX5000 modules (Ethernet) in configurations with NIC Teaming (redundancy managed by the user).

6.3.22. Redundancy Overhead

A redundant application implies on an application processing time increase, when compared to the necessary time for a non-redundant equivalent application.

This additional time happens due to cyclic synchronization services execution, described in the Cyclic Synchronization Services through NETA and NETB section, and a smaller time for the redundancy management (state machines, etc.). The total additional time due to redundancy (redundancy overhead) is estimated by MasterTool, after the redundant CPU project compiling.

ATTENTION

MasterTool calculated overhead consider an empty redundant variables forcing list.

In addition, the user must define a range for MainTask regarding:

- The necessary time to execute the main POUs (NonSkippedPrg and ActivePrg). This time usually is measured after the project development (with the redundancy additional time off)
- The time required for detection and generation of internal points events (for example, the occurance of 1000 events of analog points with deadband on the same cycle can take up to 30 ms)
- Some MainTask cycle looseness, for other CPU tasks execution (operational system, I/O PROFIBUS drivers, MODBUS, etc.). This looseness percentage can vary according to the requested performance from these other tasks. E.g. if the MODBUS communication with the SCADA system needs to allocate too much processing to reach a satisfying performance, this looseness must be increased

ATTENTION

Depending on the memory alignment, the number of bytes used in the redundancy overhead calculus might be higher than the total amount of bytes declared in the variables.

6.4. Redundant CPU Programming

6.4.1. Wizard for a New Redundant Project Creation

In order to create a new redundant project, the File/New Project command must be used and the MasterTool Standard Project selected.

Initially, the user must inform the desired name for the project and the directory where he desire to save it, as shown on figure below.

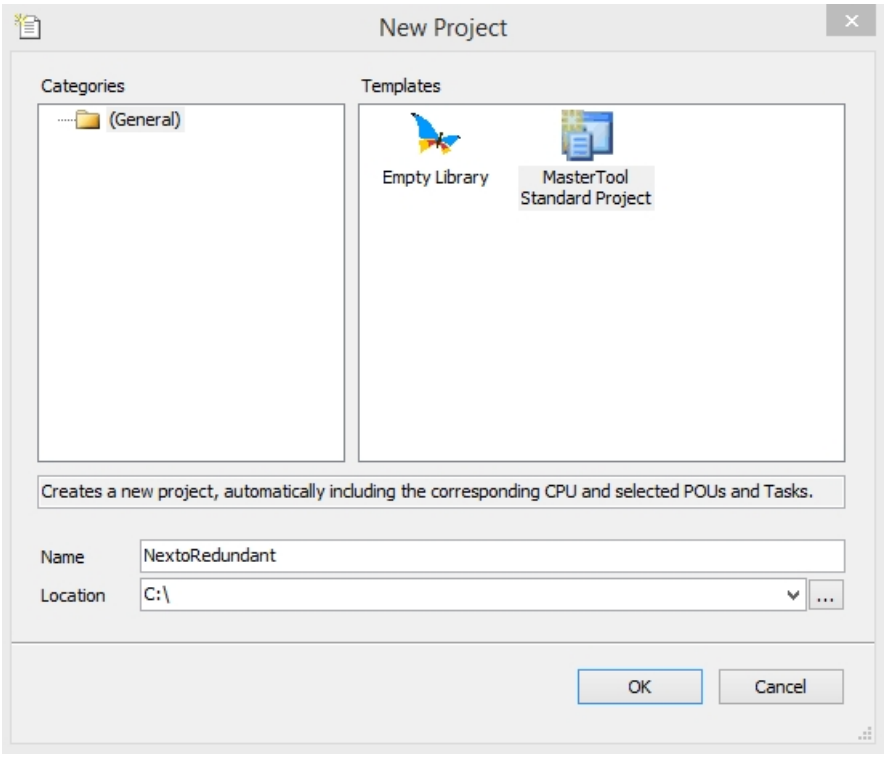


Figure 173: New Project

Next, the Wizard which generates the redundancy project run some questions for the user, regarding the desired configuration that must be answered successively.

The first point to be defined is the initial configuration for the half-cluster hardware:

- Select device category: NX3030 can be selected in two categories All Devices or in Modular Controllers
- Select the CPU model: As the redundancy is implemented only in NX3030, it must be selected by the user
- Select the rack model: There are four rack available models and the choice depends on the module quantity used in the redundancy. For MasterTool is important the rack size according to the configured networks quantity (next wizard item)
- Select the power supply model
- Select the redundancy configuration. For a redundant project is needed to choose With Redundancy option
- Select the operation mode of redundancy. In this case the options in operation are with or without redundancy panel (PX2612)
- Select if the OPC DA communication option will be enabled or not
- Select if will be used bus expansion redundancy

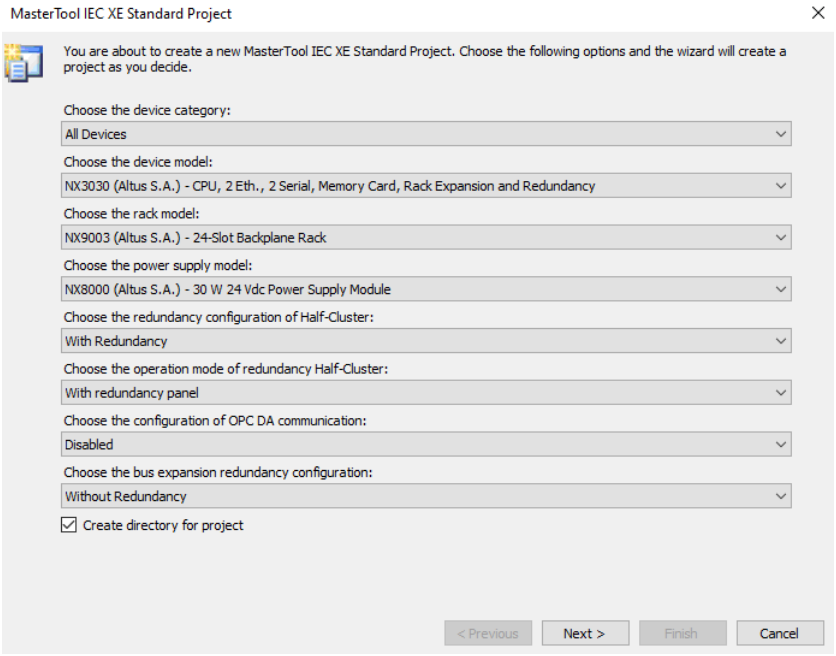


Figure 174: Hardware Initial Configuration

After, the user must define the communication networks used in the redundant application:

- Select device category: NX3030 can be selected from two categories All Devices or Modular Controllers
- Select the number of PROFIBUS networks: By the Wizard, can be created up to four PROFIBUS networks, and they can be single or redundant. It is important stress that this architecture proposed by the Wizard is typical. After that, can be created more PROFIBUS networks, respecting the maximum limit of four PROFIBUS Master modules, NX5001, in each half-cluster
- Choose the type of PROFIBUS networks:
  - There's none (no NX5001 module allocated)
  - Single Network (allocates one NX5001 module)
  - Redundant Network (allocates two NX5001 modules)
- Choose the type of Ethernet network of the CPU
  - Single Network with Failure Mode Disabled (do not generates switchover in failure case)
  - Single Network with Failure Mode Enabled (generates switchover in failure case)
  - Redundant Network with Failure Mode Disabled (operates in conjunction with the other interface and do not generates switchover in failure case)
  - Redundant Network with Failure Mode Enabled (operates in conjunction with the other interface and generates switchover in failure case)
- Choose the amount of Ethernet networks: In this case the Wizard allows the user to create up to four single networks, or up to three redundant networks, or none. It's important to stress that this is only the architecture proposed by the Wizard. After that, MasterTool allows the creation up to six networks total (three redundant maximum), always respecting the maximum limit of six Ethernet modules, NX5000, in each half-cluster.
- Select the Ethernet network type:
  - There's none (no NX5000 module allocated)
  - Single Network with Failure Mode Disabled (allocates one NX5000 and do not generates switchover in failure case)
  - Single Network with Failure Mode Enabled (allocates one NX5000 and generates switchover in failure case)
  - Redundant Network with Failure Mode Disabled (allocates two NX5000 and do not generates switchover in failure case)
  - Redundant Network with Failure Mode Enabled (allocates two NX5000 and generates switchover in failure case)

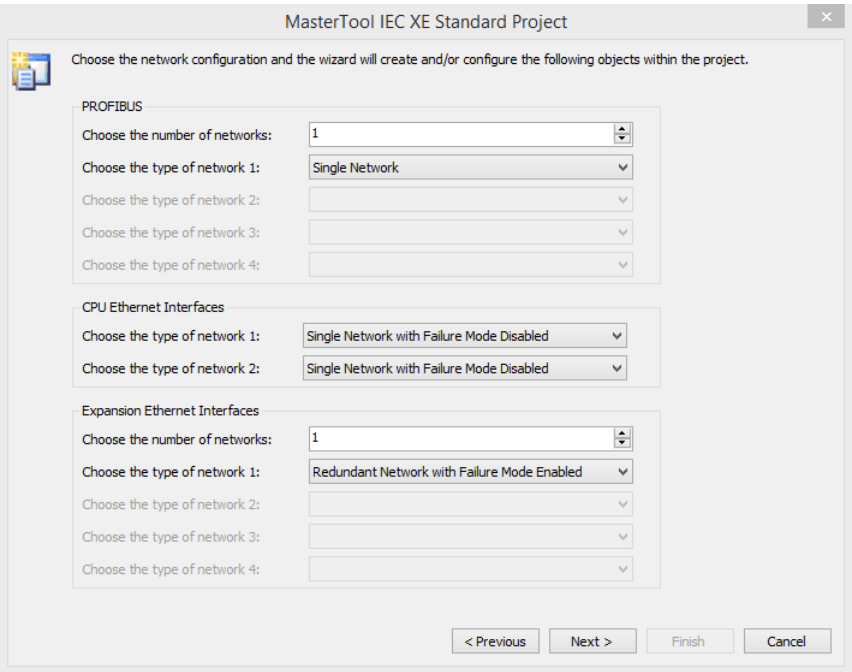


Figure 175: Communication Networks Configuration

Then the project profile and the standard language must be selected for the program creation:

- Select the project profile: It's only possible to use the Single project profile for the redundancy; hence the selection option is disabled
- Select the default language for all programs: The language selected by the user is the standard for all programs, but any other can be used for a specific POU

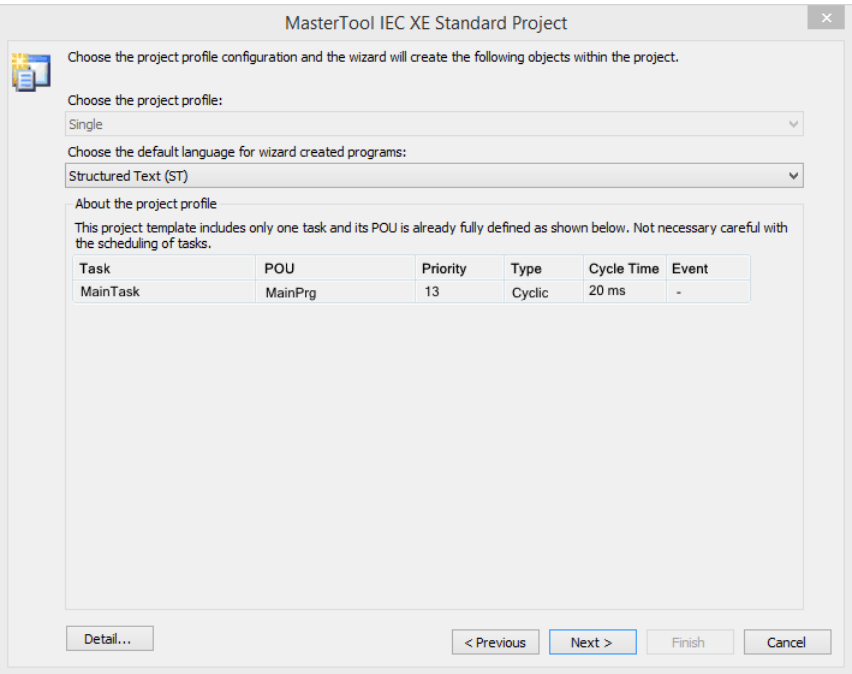


Figure 176: Project Profile and Standard Language

To finish, the user must select the program language common and associated to the redundancy:

- Program associated with MainTask (MainPrg): It must be, obligatory, in ST language, as MasterTool disables the other options
- Programs associated with redundancy Main Tasks

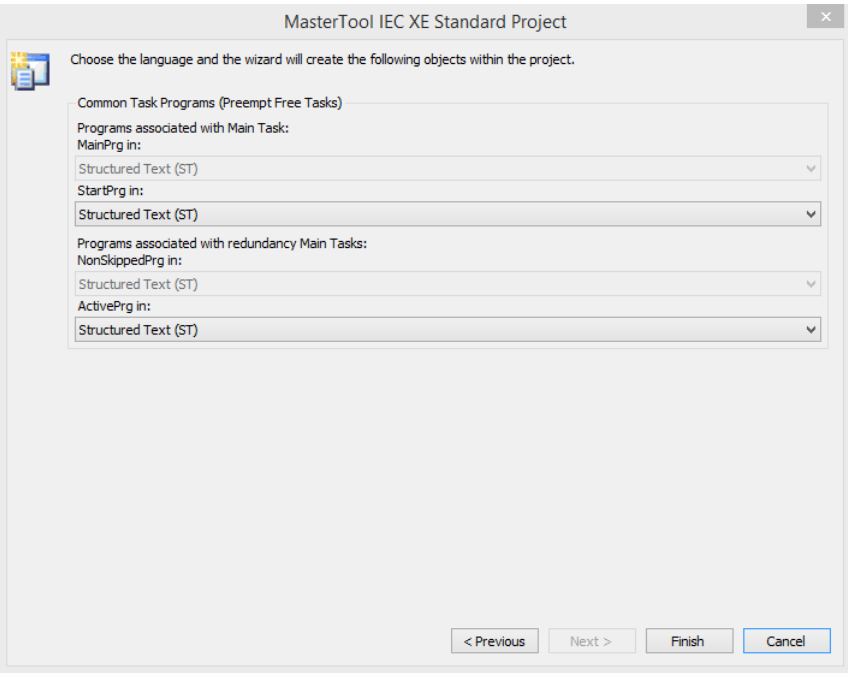


Figure 177: Specific Programs Language

ATTENTION

The ActivePrg and NonSkippedPrg POUs are created automatically, empty, in language selected on the previous questions. Other POUs which are created manually by the user can be used in any available language, except in redundant POUs which can't be written in SFC language as it uses the IEC timer as background. For further information see Limitations on a Redundant PLC Programming.

ATTENTION

The MainPrg POU will always be automatically generated in ST language, and cannot be changed by the user. This POU calls the ActivePrg (only in the Active PLC) and NonSkippedPrg (in both PLCs) POUs.

After receiving the answers for the previous questions, the Wizard generates the main project, defining a half-cluster with the following initial hardware configuration:

- Selected rack
- Power supply NX8000 (positions 0 and 1)
- NX3030 CPU (positions 2 and 3)
- NX4010 modules (positions 4 and 5) and Panel PX2612 if selected.
- After the NX4010 module, NX5001 are inserted to implement PROFIBUS network with the features previously inserted by the user
- After the NX5001 modules, NX5000 are inserted to implement Ethernet network with the features previously inserted by the user

6.4.2. Half-Clusters Configuration

The Wizard is always used to generate the first version of a redundant project. This guarantees the initial version is generated quick and correctly.

However, it's possible that some modifications are necessary in a half-cluster, such as the insertion of new NX5001 and NX5000 modules that can be executed changing the half-cluster configuration screen. The following sections present how to insert and configure the modules NX5000, NX5001 and NX4010.

Some rules and precautions must be followed for a redundant project, as described in the following sections.

6.4.2.1. Fixed Configuration in the 0 to 5 Rack Positions

In the 0 to 5 positions of the selected rack, the following modules must be always installed:

- Power supply NX8000 (positions 0)
- NX3030 CPU (positions 2)
- NX4010 module (positions 4)

These modules must not be removed from the original project generated by the Wizard.

Any different configuration in these positions results in an error displayed by MasterTool at the project compilation.

6.4.3. Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2)

6.4.3.1. IP Address Configuration

The figure below presents the CPU NX3030 NET 1 port configuration (the screen for NET 2 port configuration has a subgroup of these parameters). In order to open this screen, a double click must be executed on NET 1 or NET 2, below the CPU NX3030 in the device tree.

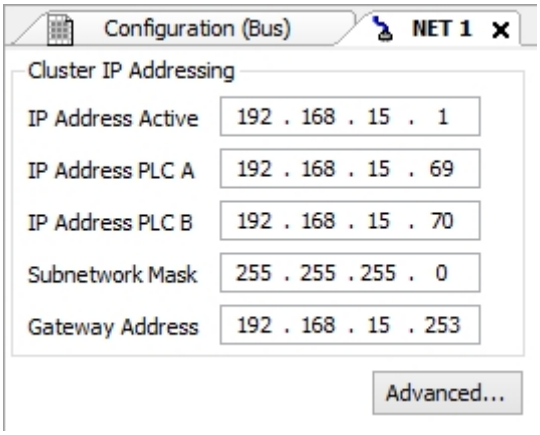


Figure 178: Ethernet NET 1 Port Parameters

Next the basic parameters of the NET 1 and NET 2 interfaces must be edited. The address has to be set according to the Active IP Change method, as described in Principles of Operation - IP Change Methods - Active IP.

ATTENTION

The NET 1 and NET 2 interfaces IP addresses, as the Gateway Address, must belong to the same subnet.

ATTENTION

The NET 2 configuration screen has the same structure as the NET 1 configuration screen, but it doesn't have the checkbox Redundancy of Communication, neither the NIC Teaming configuration parameters.

6.4.3.2. NIC Teaming between NET 1 and NET 2

The Advanced option on the NET 1 configuration screen opens a new configuration screen, which defines if NET 1 will be redundant. In case the checkbox for Redundancy of Communication is marked, the NET 1 and NET 2 interfaces form a redundant pair with NIC Teaming, as described in the Principles of Operation - Redundant Ethernet Networks with NIC Teaming section. Automatically, other parameters are enabled and must be configured:

- Period of Redundancy Test (ms): Period to transfer the communication test frame between the two NETs. It can be configured with values between 100 and 9900
- Retries of Redundancy Test: Maximum number of times the NET, which has sent the frame, will wait for an answer. It can be configured with values between 1 and 100
- Switching Period (s): Maximum time the Active NET will wait for any package. It can be configured with values between 1 and 25

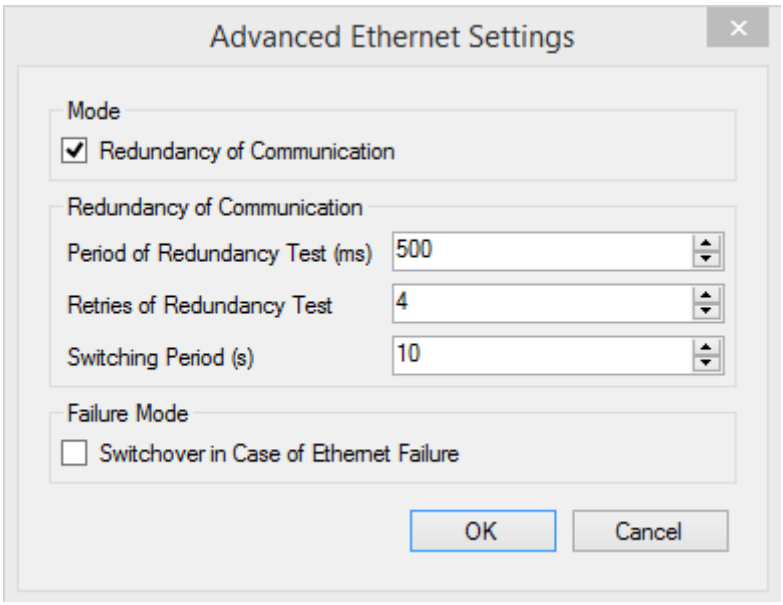


Figure 179: Ethernet Advanced Configuration

In case the answer time for the Redundancy Test reaches the Period of Test times the Number of Retries and the active interface remains for a while longer than the Switching Period without receiving any package, a switchover will occur, turning active the interface that was inactive. It is important to stress that there is a delay between the failure detection and the activation of the inactive interface, due to the time necessary to interface configuration. This delay could be up to a few dozens of milliseconds.

When one of the NETs is active, it assumes the IP address configured, and the inactive NET remains with its configured IP Address, Subnetwork Mask and Gateway Address parameters blank in the CPU diagnostics.

ATTENTION

When a Reset Origin is performed in a CPU configured with NIC Teaming enabled for local Ethernet interfaces (NET 1 and NET 2), only the last active interface before the reset will be accessible. After the reset command, the accessible interface could be viewed in the CPU's Informative and Configuration Menu.

6.4.3.3. Vital failure setting in NET 1 and NET 2

The Advanced option in the setup screen of the NET 1 and NET 2 interfaces, opens a configuration screen where in addition to enable communication redundancy is also possible to configure if the interface will generate a switchover in case of failure as described in Principles of Operation - Ethernet Interfaces Use with Vital Fault Indication.

When configured in conjunction with the NIC Teaming redundancy, failure is considered vital failure, when a fault occurs in NET 1 and NET 2 interfaces.

6.4.4. NX5001 Modules Configuration

6.4.4.1. Insertion or Removal of NX5001 modules

NX5001 modules can be inserted or removed from the half-cluster rack. To execute this operation correctly, one must be aware of the following rules:

- The number of NX5001 modules in each half-cluster may vary between zero and four
- It can be defined up to 4 simple PROFIBUS networks or 2 redundant PROFIBUS networks, respecting the limit of 4 PROFIBUS Master NX5001 modules in each half-cluster
- When a PROFIBUS network is simple, it needs a single NX5001 module in each half-cluster. When it's redundant, it needs 2 NX5001 modules in each half-cluster
- Two NX5001 modules used to form a redundant PROFIBUS network must occupy side by side positions in the rack
- The NX5001 modules quantity in the rack must be compatible with the number of existent PROFIBUS networks and with the redundancy attribute of each network, on other words:
  - 0 x NX5001: No PROFIBUS network
  - 1 x NX5001: One simple PROFIBUS network
  - 2 x NX5001: In this case there are two options:
    - Two simple PROFIBUS network
    - One redundant PROFIBUS network
  - 3 x NX5001: In this case there are two options:
    - Three simple PROFIBUS networks
    - One redundant PROFIBUS network and one simple PROFIBUS network
  - 4 x NX5001: In this case there are three options:
    - Four simple PROFIBUS networks
    - One redundant PROFIBUS network and two simple PROFIBUS networks
    - Two redundant PROFIBUS networks

After inserting or removing the NX5001 modules, the configuration of the NX5001 modules remaining in the rack must be checked.

6.4.4.2. NX5001 Modules Parameters Adjust

Each NX5001 module used in a simple PROFIBUS network, or each redundant pair of NX5001 used in a redundant PROFIBUS network, has the following parameters to be adjusted.

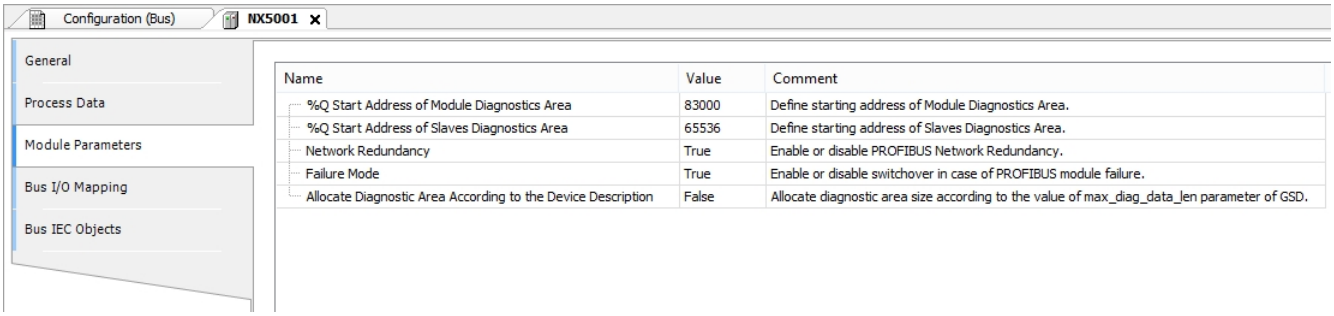


Figure 180: NX5001 Redundancy Parameters

For grouping two NX5001 modules in a redundant PROFIBUS network, a double click must be executed on an ungrouped NX5001 module which has another ungrouped NX5001 module at its right in the rack. Next the parameter Network Redundancy, available at the tab Module Parameters, must be marked as TRUE, as shown on the Figure 180. In order to ungroup it, the same procedure must be followed, but marking the parameter as FALSE. If this parameter is marked as TRUE, the DP parameters and the NX5001 parameters at its right are blocked for edition.

ATTENTION

In case of redundant networks, only the parameters of the NX5001 to the far left on the bus must be adjusted, while the NX5001 at the right remain blocked for edition. Some network parameters are identical to the other network while others are calculated automatically from network parameters of the left NX5001.

It's recommended for the configured address for a NX5001 master in a redundant PLC to be 2, as the master NX5001 address in the Non-Active PLC is decremented one unit, thus the NX5001 master address results 1.

Besides that, it's important to remember:

- The addresses from 3 to 125 are usually used for PROFIBUS slaves
- The 0 address is frequently used for device configuration and diagnostics
- The address 1 is reserved to be taken, dynamically, by the PROFIBUS master in the Non-Active PLC (PROFIBUS master in passive mode)
- The 126 address is frequently used for slave devices when comes from the manufacturer
- The 127 address is used for broadcast frames

In the next project compilation, MasterTool check the possible errors the user may have made at inserting or removing NX5001 modules manually.

Important to note that during the execution of a project previously configured with redundant NX5001 modules, bit 0 Command (Channel Enable Interface %QXn.0 at Bus I/O Mapping tab) is handled by the redundant application. The interfaces must remain qualified throughout the program. Thus, a command run by the user to disable an interface will not run the way it's expected. For example, if an interface has the status of this bit changed from TRUE to FALSE on an active CPU, this will not be interpreted as a failure that would take the CPU Active for the Inactive state. In this case, the CPU will remain in Active and the other CPU that will go to the Inactive state. For these reasons, this command bit should not be manipulated by the user in a redundant application.

For further information regarding PROFIBUS networks configuration, see PROFIBUS-DP NX5001 Utilization Manual.

6.4.4.3. PROFIBUS Remotes Configuration

To configure PROFIBUS remotes under a NX5001 master, the PROFIBUS-DP NX5001 Master Utilization Manual must be consulted, together with the following manuals:

- Ponto Series Utilization Manual
- PROFIBUS PO5063V1 Head and Redundant PROFIBUS PO5063V5 Head Utilization Manual
- PROFIBUS PO5064 Head and Redundant PROFIBUS PO5065 Head Utilization Manual
- HART over PROFIBUS Network Utilization Manual

For a redundant system we must pay attention to the configuration of the watchdog parameter from the PROFIBUS remote. In case that, in the remote configuration screen, the Watchdog control checkbox is checked, the Time field needs to be correctly configured. There are two options to configure the Time and we must use the bigger time between:

- WT $\geq$ I x 2 + 500ms; and
- WT $\geq$ I x 3;

Where WT is the watchdog time and I is the MainTask configured interval.

Figure 181: Watchdog Configuration of a PROFIBUS Remote

6.4.5. NX5000 Modules Configuration

6.4.5.1. NX5000 Modules Insertion or Removal

NX5000 modules can be inserted or removed from the half-cluster rack. To execute this operation correctly, one must be aware that the number of NX5000 modules in each half-cluster can vary between zero and six. Care must be taken to the fact that modules which form a redundant NIC Teaming pair must be inserted in side by side positions in the rack.

In the next project compilation, MasterTool check the possible errors the user may have committed at inserting or removing NX5000 modules manually. For instance, if the user inserted more than 6 NX5000 modules, an error occurs.

The interface of each module will be identified as NET 1, as they are identified physically on the product. In case the user adds manually NX5000 modules in the bus, the identification occurs the same way as the Wizard.

After inserting or removing the NX5000 modules, the configuration of the NX5000 modules remaining in the rack must be checked.

6.4.5.2. NX5000 Modules Configuration

For each NX5000 module in a redundant PLC, the address parameters must be adjusted as described in the Principles of Operation - IP Change Methods section, which can be accessed through a double click on the NET 1 interface, below each NX5000 module placed on the devices tree.

ATTENTION

In case two consecutive modules form a redundant NIC Teaming pair, only the basic parameters of the left NX5000 should be edited, the right NX5000's parameters edition will be blocked.

6.4.5.3. NX5000 Modules Grouping with NIC Teaming Redundancy

The NX5000 modules, as the CPU NX3030 and NX3020 NET 1 interface, present a screen of advanced configuration which defines if the module forms a redundant NIC Teaming pair with the module at its right. The configuration is made as described in the NIC Teaming between NET 1 and NET 2.

To group two NX5000 modules with a redundant pair, the following conditions must be true:

- Both NX5000 modules must be inserted in close positions in the rack.

At doing this the right module has its parameters edition blocked and the left module parameters turn to be the same to both modules. Unmarking the checkbox Redundancy of Communication at the left module causes the modules' separation, making them behaves as individual modules without NIC Teaming redundancy again.

6.4.5.3.1. Failure Vital Setting

The NX5000 modules as well as the NET 1 and NET 2 interfaces allow you to configure if the interface will generate a switchover in case of failure, as described in Principles of Operation - Ethernet Interfaces Use with Vital Fault Indication.

When configured in conjunction with the NIC Teaming redundancy vital failure will be considered when failure occurs in both modules of the redundant pair.

6.4.6. NX4010 Redundancy Configuration

The configuration regarding the %I, %Q and %M redundant variables can be accessed through a double click on the NX4010 module, following the selection of the tab Redundancy Parameters.

To understand these parameters the sections Redundant and Non-redundant %I Variables, Redundant and Non-redundant %Q Variables and Redundant and Non-redundant %M Variables must be read.

The following parameters must be configured:

Configuration	Description	Default	Options
	Memory (%M)		
Redundancy %M Memory Offset	Redundant %M memory initial address	0	0 (disabled)
Redundancy %M Memory Length	Redundant %M memory size	0	0 to 65536
	Memory (%I)		
Redundancy %I Memory Offset	Redundant %I memory initial address	0	0 (disabled)
Redundancy %I Memory Length	Redundant %I memory size	16384	0 to 81920
	Memory (%Q)		
Redundancy %Q Memory Offset Reserved For I/O Drivers	%Q redundant memory offset reserved for I/O drivers initial address	0	0 (disabled)
Redundancy %Q Memory Length Reserved For I/O Drivers	%Q redundant memory offset reserved for I/O drivers size	16384	0 to 81920
Redundancy %Q Memory Offset Reserved For Diagnostics	%Q redundant memory offset reserved for diagnostics initial address	65536	0 to 81919
Redundancy %Q Memory Length Reserved For Diagnostics	%Q redundant memory offset reserved for diagnostics size	16384	0 to 81920

Table 203: NX4010 parameters

6.4.7. I/O Drivers Configuration

The configuration of I/O drivers, at first, isn't different in relation to a non-redundant CPU.

What can be observed is that some I/O drivers have commands which allow its use in a redundant CPU, but it doesn't imply in configuration differences. These commands, normally, must be executed in the NonSkippedPrg program. E.g. a MODBUS RTU master driver in a RS-485 serial network must be disabled in a non-Active CPU using the code inserted by the user in NonSkippedPrg. More information regarding administration of MODBUS driver in a redundant system can be found in the MODBUS Instances Managing in Redundant System section.

In the case of PROFIBUS network, there are also special different commands for the CPUs in Active and Non-Active states. In this case, however, the redundancy management executes such commands automatically, without any user management.

To configure PROFIBUS I/O remotes, including remotes and I/O modules, see NX5001 Modules Configuration section from this manual.

6.4.8. MainTask Configuration

The configuration screen associated to the only task of a redundant CPU, called MainTask, which is cyclic, can be accessed through a click on the MainTask in the Device Tree.

Two parameters must be adjusted on this screen:

- MainTask Interval
- Watchdog Time

Furthermore, the screen shows an estimative of the necessary time to manage the redundancy, calculated by MasterTool. Such estimative is only reliable after the project is complete, with all POUs developed and redundant memory areas defined. Several considerations must be taken in order to adjust correctly the MainTask cycle time:

- The interval time must be sufficiently low to allow the proper process control, taken in account all control feedback times
- The interval time must be high enough for allowing, at least, the sum of the following times:
  - The NonSkippedPrg and ActivePrg POUs maximum execution time, together

  - The necessary time to manage the redundancy (redundancy overhead)
- Besides this, the interval time must have an additional looseness necessary for the other processes execution times (PROFIBUS communication, Ethernet communication with SCADA systems, etc...)

MasterTool has conditions of calculating the necessary time for redundancy management (redundancy overhead), after the project is finished (all developed POUs and redundant memory areas defined).

Regarding the NonSkippedPrg and ActivePrg POUs execution maximum time, they are possible to be measured after these POUs are already developed. Initially, MasterTool estimates 10 ms for these two POUs maximum time, together, but the user must revise this field afterwards, when measuring using the final project.

After each compilation, MasterTool sums the redundancy overhead calculated with the parameter which informs the POU times (NonSkippedPrg and ActivePrg), and verifies if the minimum looseness parameterized is being obeyed.

E.g.:

- Parameters configured in the MainTask screen:
  - MainTask Interval: 100 ms
  - POUs NonSkippedPrg + ActivePrg estimated time: 10 ms
  - Minimum tolerance: 30%
- Calculated Overhead for redundancy: 50 ms

In this case, the total time used is 60 ms (10 ms + 50 ms), which consists in 60% of the MainTask cycle (100 ms). This way, the maximum looseness is 40% and the minimum looseness of 30% is being respected.

ATTENTION

A compilation error is produced in case the minimum looseness isn't respected, if it is configured in the CPU Project Parameters.

ATTENTION

The compilation being successful or not, MasterTool informs the calculated looseness and the redundancy overhead predicted on the message window.

6.4.8.1. ActivePrg Program

In this POU the user must create the main application, responsible for its process control. This POU is called by the main POU (MainPrg), being executed only in the Active CPU.

The user can also create additional POUs (program, function or function block), and call or instance them inside the ActivePrg POU, in order to structure his program. It's possible to call functions and instance function blocks defined in libraries, too.

It must be remembered that all symbolic variables defined in the ActivePrg POU, as the instances of function blocks, are redundant variables. Symbolic variables defined in additional POUs from the program type which are called inside the ActivePrg, are also redundant variables.

ATTENTION

Variables from the type VAR_TEMP must not be used in the redundant program.

6.4.8.2. NonSkippedPrg Program

This POU is used for controls which must be executed in both CPUs (PLCA and PLCB), independent on the redundancy state. This POU is also called by the main POU (MainPrg).

It must be remembered that all symbolic variables defined in the NonSkippedPrg POU, as well as the function blocks instances, are non-redundant variables. The user must create additional POUs (program, function or function block), and call or instance them inside the NonSkippedPrg POU, in order to structure his program. It's possible to call functions and instance function blocks defined in libraries, too.

ATTENTION

It must be avoided to call additional POUs from the program type inside the NonSkippedPrg, as symbolic variables declared in this type of POU are redundant, and inside the NonSkippedPrg it's normally desirable non-redundant variables. Usually the NonSkippedPrg code is small and doesn't need to call additional POUs from the program type for its structure. If the NonSkippedPrg structure is needed, function blocks or functions must be used.

Typical examples of controls executed in the NonSkippedPrg are the following:

- To create a compact diagnostics structure (%Q) to be reported to a SCADA system, from a complete diagnostics structure, where many diagnostics are not interesting for the SCADA system. These diagnostics can be extracted from data structures as RedDgnLoc, RedDgnRem, RedUsrLoc, RedUsrRem, etc.
- To copy commands received from a SCADA system for the respective data structure RedCmdLoc fields, and interconnect these commands if necessary
- To manage switchovers controlled by the user, in case of not vital failures such as the communication with a SCADA system or with a MODBUS device
- Enable and disable some specific I/O drivers, depending on the redundancy state (Active or Non-Active). E.g. a MODBUS RTU master driver in a RS-485 bus must be disabled in the Non-Active CPU. For further information see MODBUS Instances Managing in Redundant System section

ATTENTION

It's not recommended to use function blocks TOF_RET, TON_RET, TOF and TON in the NonSkippedPrg program. See Limitations on a Redundant PLC Programming.

6.4.9. Redundancy Configuration Object

This object, located in the device tree, is automatically created by the Wizard. It is used to determine which POUs and GVLs are redundant, and therefore synchronized between PLCs. By default, POUs and GVLs created by the user are marked as redundant, leaving the option to the user to reverse the marking when needed.

ATTENTION

PV, PIDControl and PidRetainGVL objects can't be individually marked. In case of need to modification, the *Select all* option must be marked.

6.4.10. GVL Module_Diagnostics

This special GVL is created and filled automatic by the Wizard and can't be modified by the user.

System diagnostics and commands, including redundancy data structure (RedDgnLoc, RedDgnRem, RedCmdLoc, RedCmdRem), are placed within direct representation special variables %Q or %I.

The *Module_Diagnostics* GVL has many sentences with the AT keyword to define symbolic names for these diagnostics and commands. This way, when the user needs to reference these variables, he can use a symbolic name instead a numeric reference.

6.4.11. GVLs with Redundant Symbolic Variables

The user can create other GVLs different from the previously listed, in order to declare redundant symbolic variables. For that, after the GVL creation, it's necessary to mark it in the object *Redundancy Configuration*, in the project devices tree. By default, all GVLs created by the user are, initially, redundant.

ATTENTION

For good practice it's recommended to avoid the AT directive use in GVLs which have redundant symbolic variables declaration to prevent variable mapping in non-redundant areas.

6.4.12. POUs from the Program Type with Redundant Symbolic Variables

The user can declare redundant symbolic variables in POUs from the program type, with exception of the NonSkippedPrg POU where the symbolic variables declared are considered redundant.

In order to define a new POU as redundant, it must be marked in the *Redundancy Configuration* object after its creation, in the project devices tree. By default, all POUs created by the user are, initially, redundant.

ATTENTION

For good practice it's recommended to avoid the AT directive use in POUs which have redundant symbolic variables declaration to prevent variable mapping in non-redundant areas.

6.4.13. Breakpoints Utilization in Redundant Systems

For redundant systems it's recommended to use breakpoints only in the Active half-cluster, with the other half-cluster deactivated. If not, when the application execution reaches a breakpoint, the Stand-by breakpoint will take over the Active state, switching off the Active PLC.

6.4.14. MODBUS Instances Managing in Redundant System

In case the vital fault of the Ethernet interfaces is disabled or in the case of MODBUS instances Server, MODBUS instances are independent of redundancy and, therefore, must be managed in the application, being at the user's discretion which instances should be enabled/disabled when a PLC enters a Non-Active state. When the vital fault is enabled for Ethernet ports with MODBUS Client, it is not necessary to implement additional code to control the switch-over.

The example below, inserted in a NonSkippedPrg program, executes the verification of the PLC current state and in case it's in Non-Active state, disables the MODBUS RTU master and slave instances and the MODBUS Ethernet Server instance:

```
VAR
eRedStateLocal : REDUNDANCY_STATE;
eRedStateLocal_old : REDUNDANCY_STATE;
END_VAR

// Local PLC current state reading
eRedStateLocal := DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.eRedState;

// Has the local PLC state changed?
IF eRedStateLocal <> eRedStateLocal_old THEN
IF eRedStateLocal = REDUNDANCY_STATE.ACTIVE THEN
// The local PLC has entered the Active state
Diagnostics.DG_MODBUS_RTU_Slave.tCommand.bRestart := TRUE;
Diagnostics.DG_MODBUS_RTU_Master.tCommand.bRestart := TRUE;
Diagnostics.DG_MODBUS_Server.tCommand.bRestart := TRUE;
ELSE
// The local PLC has entered the Non-Active state
Diagnostics.DG_MODBUS_RTU_Slave.tCommand.bStop := TRUE;
Diagnostics.DG_MODBUS_RTU_Master.tCommand.bStop := TRUE;
Diagnostics.DG_MODBUS_Server.tCommand.bStop := TRUE;
END_IF
// Saves the last state of the local PLC
eRedStateLocal_old:= eRedStateLocal;
END_IF
```

6.4.15. Limitations on a Redundant PLC Programming

On a redundant PLC there are some limitations regarding its half-cluster programming. These limitations are treated in the subsections below.

6.4.15.1. Limitations in Redundant GVLs and POUs

In a redundant GVL or a POU from the program type the following limitations must be respected for a correct functioning of the half-clusters:

- Do not use variables from the type VAR_TEMP
- Do not mix variable types (VAR, VAR RETAIN, VAR PERSISTENT, VAR CONSTANT, etc...). Only one type must be used in each GVL or POU
- Do not mix symbolic variables declaration with ATs in the GVLs. Separate GVLs must be created where in one the AT variables will be declared and in another, the symbolic variables
- Do not store a variable address in a redundant variable (use a redundant variable as a pointer), as the variable addresses may be different in the PLCA and PLCB
- Do not use the function blocks for RTC reading and writing in redundant POUs. More details can be found on the section RTC Clock

6.4.15.2. Non-redundant Program Limitations (NonSkippedPrg)

In a POU from the program type which aren't redundant, the case of a NonSkippedPrg POU, the following limitations must be respected for a correct functioning of the half-clusters:

- The traditional function blocks TON and TOF can't be used as they use the IEC timer. When the Stand-by PLC goes to Active state (with the other half-cluster coming out of Active state), the IEC timer is synchronized, causing a discontinuity in the timer value. The function blocks TON_NR and TOF_NR must be used instead, available in the *NextoStandard* library. See Non-Redundant Timer
- POUs from the program type written in the SFC language (Sequential Function Chart) must not be used, as they use the IEC timer for transition timing
- Do not mix symbolic variables declaration with ATs in the GVLs. Separate GVLs must be created where in one the AT variables will be declared and in another, the symbolic variables

6.4.16. Getting the Redundancy State of a Half-Cluster

It is possible to verify the redundancy state of a half-cluster in the Redundancy Diagnostics Structure:

```
VAR
eRedStateLocal : REDUNDANCY_STATE;
END_VAR

eRedStateLocal := DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.eRedState;
```

This way, the user can control a program logic that depends on redundancy state of the PLC.

6.4.17. Reading Non-Redundant Diagnostics

A redundant project, besides present redundant diagnostics (Redundancy Diagnostics Structure or the diagnostics from a PROFIBUS remote), presents also non-redundant diagnostics (diagnostics from the modules NX5000, NX5001, NX3030, etc.). These non-redundant diagnostics could be invalid and must not be considered at the first instants in Active state, as they aren't synchronized with the other PLC (the diagnostic state when the remote PLC was active is unknown). Therefore, these diagnostics must be ignored during the first moments in Active state, until they have valid values. Typically the time during which the diagnostics should not be considered is 5 s.

The example below shows how to not consider the diagnostics *bSlaveNotPresent* and *bPbusCommFail* from the NX5000 PROFIBUS Master module.

Logic in *NonSkippedPrg*:

```
PROGRAM NonSkippedPrg
VAR
TON_DiagEnable : TON_NR;
bDiagEnable : BOOL;
bIsActiveState : BOOL;
bIsActiveState_old : BOOL;
END_VAR

bIsActiveState := (DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.eRedState =
   REDUNDANCY_STATE.ACTIVE);
TON_DiagEnable(IN:= (bIsActiveState = bIsActiveState_old), PT:= T#5S, Q=>
   bDiagEnable);
bIsActiveState_old := bIsActiveState;

Logic in ActivePrg:
IF NonSkippedPrg.bDiagEnable THEN
IF DG_NX5001.tGeneral.bSlaveNotPresent OR DG_NX5001.tGeneral.bPbusCommFail THEN
//Actions executed when the diagnostics are active
END_IF
END_IF
```

6.5. Redundant CPU Program Downloading

The Redundant CPU Programming section described issues related to the development of a project for a redundant CPU with NX3030 CPU.

In this section, many methods and steps to download this project in a redundant CPU are described, considering situations such as:

- Downloading the project in a brand new NX3030 CPU or in a CPU with an unknown project
- Online modifications downloading
- Offline modifications downloading with the process control interruption, during a programmed process stopping
- Offline modifications downloading without the process control interruption, using redundancy features

6.5.1. Initial Downloading of a Redundant Project

This section describes the necessary steps to run the first download of a redundant project in a NX3030 CPU. This is necessary, for instance, for a brand new CPU recently manufactured, or for a CPU that has an unknown project.

ATTENTION

The following steps must be executed for both half-clusters (PLCA and PLCB) which compose a redundant CPU. First all steps must be executed for one CPU and then for the other.

6.5.1.1. First Step – IP Address Discovering for MasterTool Connection

The first step is to discover the IP address from the NET 1 channel in this CPU, for MasterTool connection.

This must be done through NX3030 CPU display and button, as described in the CPU's Informative and Configuration Menu section. The NETWORK menu informs the IP address which can be used for MasterTool connection.

6.5.1.2. Second Step – Verifying IP Addresses Conflict

Before executing the third step, one must be sure there's no other equipment with the same IP address connected to the network, discovered in the first step. This can be discovered, for instance, disconnecting the CPU from the network and executing a "ping" in its IP address. As the CPU is disconnected from the network, the "ping" function must fail. If not, there's equipment with the same IP address.

In case the IP address is already being used by equipment in the network, the third step must be executed, and some of the following steps too, using a crossover cable to connect MasterTool to the CPU, avoiding IP addresses conflict. In one of the following cases, at downloading the project in the CPU, the definitive IP addresses are updated in it (see Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2) section).

6.5.1.3. Third Step – Preparing MasterTool Connection (Set Active Path)

The third step consists in double-clicking on the Device (NX3030 PLC) in the Device Tree, getting in the tab "*Communication Settings*", clicking on the Gateway, and pressing the "*Scan Network*" button to list all CPUs detected by MasterTool in the network.

At this moment, a CPU whose identification has the IP address found in the first step is supposed to appear. In case the user has changed the network CPU name previously, this name will be visualized. MasterTool Connection with a NX3030 CPU from a Redundant PLC section describes with more details the possible identifications which can be observed on this list. Anyhow, all possible identification has a field showing the IP address or part of it. For instance, the bytes between square brackets form the CPU address. The right byte inside the brackets, indicate the IP address end in hexadecimal. If the bytes form the address [0010], this means the byte with value "10" indicates that the CPU IP address end is xxx.xxx.xxx.16. Next, the CPU in the list must be clicked and the "*Set Active Path*" button pressed. This done, the selected CPU must appear stressed on the list, indicating MasterTool is prepared to connect to this CPU.

6.5.1.4. Forth Step – Identifying the NX3030 CPU and Verifying the CPU Display

The forth step consists in identifying the half-cluster as PLCA or PLCB. This is made through the *Online* / *Redundancy Configuration* menu:

Next, the combo-box "*PLC Identification*" allows selecting one out of the three following options:

- PLC A
- PLC B
- Non-Redundant

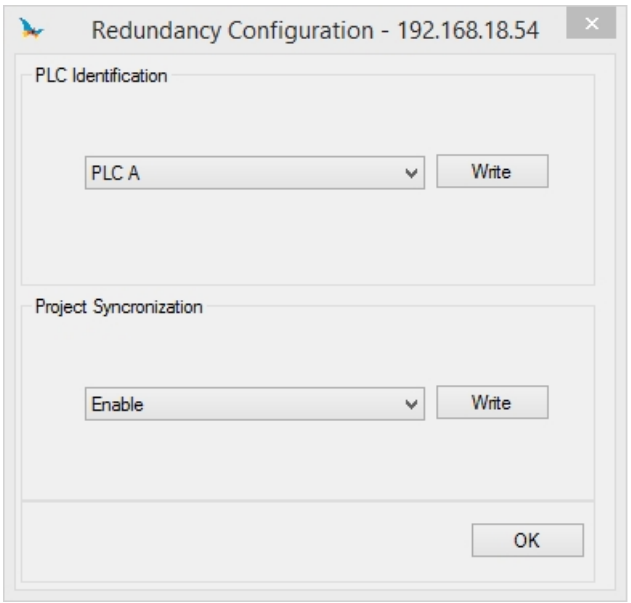


Figure 182: PLC Identification

In case of a redundant CPU, the user must select PLCA or PLCB. After selecting the desired option, the "*Write*" button correspondent to this combo-box must be pressed. MasterTool returns a message warning that the CPU will be restarted and

waits for the user to confirm the action. Then a message indicating command success or failure will appear. If there's success the CPU will be restarted.

ATTENTION

The NX3030 CPU can't be in Run mode when this command is executed. Before executing this command, the user must put the CPU to Stop mode. In case the CPU is in Run mode, the command isn't executed and MasterTool warns the command has failed.

Just after executing the identification command with success, it can be observed that the selected identification appears on the Redundancy Diagnostics on NX3030 CPU Graphic Display.

The CPU identification is also available in an internal diagnostic (DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.ePLC_ID). This diagnostic is updated from the non-volatile memory each MainTask cycle, so it's necessary for the CPU to go back to Run mode to update it. The codes returned by the diagnostics and its respective limitations are listed below:

- Non-Redundant: 0
- PLCA: 2
- PLCB: 3

The CPU identification isn't part of the redundant project developed with MasterTool. Such identification is only in a CPU non-volatile memory area, which can be modified using MasterTool.

CAUTION

The redundancy doesn't work properly in case one of the CPUs isn't identified as PLCA and the other PLCB, when a process control interruption may occur. In case one NX3030 CPU must be replaced (e.g. after a damage), the new CPU must be previously identified with the same identification of the damaged one. The CPU display must be used to verify if both CPUs are correctly identified.

6.5.1.5. Fifth Step – Redundant Project Downloading

This step describes the redundant project downloading in the CPU. This project must be prepared according to the Redundant CPU Programming section.

A simple project (basic) can be prepared following, at least, the next subsections presented in this section:

- Wizard for a New Redundant Project Creation
- Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2)

Obviously, it's also possible to build a complete project and only later download it in the PLCA and PLCB, for instance, in case these CPUs hardware aren't available during the project developing in MasterTool.

The first download of a redundant project in a CPU, previously identified as PLCA or PLCB, still must be done using the IP address discovered in the first step, and selecting the third step of this procedure. The project download is run through the *Online* / *Login* menu.

ATTENTION

Inside the developed project using MasterTool and downloaded in the CPU in this step, were defined new IP addresses for the NET 1 interface in the PLCA and PLCB (IP Address PLC A and IP Address PLC B), as well as an IP address for the NET 1 interface in the Active CPU (IP Address Active) – see Ethernet Ports Configuration in the CPU NX3030 (NET 1 and NET 2) section.
Therefore, after this first download, the IP address discovered in the first step of this procedure usually isn't valid anymore. This IP Address change in NET 1 causes a connection loss between MasterTool and the CPU, which is showed on the screen.
For further details regarding MasterTool reconnection, see MasterTool Connection with a NX3030 CPU from a Redundant PLC section.

6.5.2. MasterTool Connection with a NX3030 CPU from a Redundant PLC

After executing the procedure described in the Initial Downloading of a Redundant Project section in both PLCs (PLCA and PLCB), MasterTool connection, through the NET 1 interface from NX3030 CPU can be made through one of the following addresses:

- IP Address PLC A: NET 1 address exclusive for PLCA
- IP Address PLC B: NET 1 address exclusive for PLCB

Independent from the PLC state, MasterTool can only connect to it using the PLC exclusive address, configured in IP Address PLC X. But in case the PLC is in Active state, all other services can connect to it either by the IP Address PLC X or by the IP Address Active.

To connect to a specific PLC, at first a double-click must be done on the *Device* (NX3030) in the Device Tree, go into "*Communication Settings*" tab, click on the *Gateway* and press "*Scan Network*" button to list all PLCs detected by MasterTool in the network.

On this list it's possible to find the following standard identifications, in case the PLC name on the network hasn't been changed previously by the user:

- NX3030_<IP address>_PLCA: identification related to the PLCA. In this case, the field <IP address> must be the same as the IP Address PLC A configured in the project.
- NX3030_<IP address>_PLCB: identification related to the PLCB. In this case, the field <IP address> must be the same as the IP Address PLC B configured in the project.

Next, the PLC which MasterTool is to connect must be selected from the list and the button "*Set Active Path*" must be pressed. Then, at executing the command from the *Online* / *Login* menu, MasterTool connects to this PLC.

ATTENTION

MasterTool can only connect to one PLC at a time. To connect to several PLCs, multiple instances must be open in MasterTool, when care must be taken to open the correct project in each instance.

6.5.3. Modification Download in a Redundant Project

After both PLCs (PLCA and PLCB) from the redundant pair had its initial program already downloaded, as described in the Initial Downloading of a Redundant Project section, it's possible to download successive changes in the project, when such changes are necessary.

MasterTool connection to the PLCs responsible for the modifications download must be executed as described in MasterTool Connection with a NX3030 CPU from a Redundant PLC section. In this section it is explained how it's possible to connect to a specific PLC (PLCA or PLCB), to the Active PLC or to the Non-Active PLC.

Usually the modifications must be downloaded to the Active PLC and next automatically synchronized with the Non-Active PLC, through synchronism channels NETA/NETB. Therefore, MasterTool normally must use the Active PLC exclusive IP address (IP Address PLC X) to connect to NET 1 channel from the NX3030 CPU in the Active PLC. In order to verify which PLC is in Active state, the same step described in Initial Downloading of a Redundant Project - Forth Step – Identifying the NX3030 CPU and Verifying the CPU Display can be followed.

ATTENTION

To download a project in the Non-Active PLC is usually useless as the project automatic synchronization (Active to Non-Active PLC) would cancel the effect of this download. However, there are special situations when the project synchronization must be disabled temporarily, being possible and useful to download a different project in the Non-Active PLC. These special situations are discussed in the Exploring the Redundancy for Offline downloading of Modifications without Interruption of the Process control section.

6.5.4. Offline and Online Modifications Download

Project modifications may be downloaded offline or online.

Offline downloads require the PLC, where the downloaded is supposed to be executed, stopping. On the other hand, online downloads allow the PLC to continue executing its application while the modification is downloaded.

Some modification types require offline download and can't be executed online in the PLC where MasterTool is connected. In this case, there are two options:

- To interrupt the process control, executing the procedure described in the Offline Download of Modifications with Process Control Interruption section
- Use the PLC and the PROFIBUS networks redundancy in order to avoid interruption of the process control, even with the necessity to execute offline downloads in each half-cluster (PLCA or PLCB). A procedure to reach this objective is described in the Exploring the Redundancy for Offline downloading of Modifications without Interruption of the Process control section

6.5.4.1. Modifications which Demand Offline Download and the Interruption of the Process Control

The following modifications in a project will make it impossible to be downloaded in a redundant system with no interruption of the process control:

- Modifications in the redundant memory areas (changes in the *Redundancy Parameters* from the module NX4010)

> **ATTENTION**
>
> Will not be possible to change the size of redundant memory areas without the interrupt of the process control. Thus, these areas must be carefully planned and previously configured.

6.5.4.2. Modifications which Demand Offline Download

The following modifications demand offline downloads in the PLC where MasterTool is connected:

- To add or remove devices from the device tree, such as:
  - Modules in the main rack (NX5001 PROFIBUS masters, NX5000 Ethernet interfaces, etc.)
  - Remotes in PROFIBUS networks
  - I/O modules in remotes in PROFIBUS networks
  - MODBUS instances
- To modify parameters inside devices from the device tree, such as:
  - IP addresses and other Ethernet interfaces parameters
  - PROFIBUS master parameters
  - PROFIBUS remotes parameters
  - I/O modules parameters inside PROFIBUS remotes
- To modify a task's period
- Project update due to MasterTool IEC XE programmer Update

6.5.4.3. Modifications which Allow Online Download

A priori, the modifications not mentioned in the sections Modifications which Demand Offline Download and the Interruption of the Process Control and Modifications which Demand Offline Download, allows online download.

Even this way, the modifications which allow online download in the PLC where MasterTool is connected are listed below. These modifications are valid for variables, POUs and GVLs, redundant or not:

- To add POUs from the program type, if these POUs don't need to be associated to any task
- To remove POUs from the program type, if these POUs aren't associated to any task
- To add or remove POUs from the function or function block type
- To modify the code of any type of POU (program, function or function block)
- To add or remove symbolic variables in any POU type (program, function or function block)
- To add or remove instances of function blocks in POUs from the program or function block type
- To add or remove GVLs
- To add or remove symbolic variables or instances of function blocks in GVLs

6.5.5. Online Download of Modifications

In the Offline and Online Modifications Download section, modifications which demand offline download were described, along with the ones that allow online download.

An online change must be made by connecting the MasterTool to the NET 1 channel of the active CPU, using its unique IP address. Before version 2.01 of the MasterTool IEC XE, it was necessary that the user selected the "*Create Boot Application*" option in the *Online* menu, after sending the application for the changes to be sent to the non-volatile memory area of the CPU and could be synchronized. From version 2.01 this operation is no longer needed. After sending the application the send operation for non-volatile memory is performed automatically.

ATTENTION

It's important to remember that online modifications, without the option mentioned previously selected, will be lost in case of a Reset Warm or a CPU switch off.

ATTENTION

An online change in the declaration of retain variables of the application (adding or removing variables) followed by a drop in the power PLC before "*Create Boot Application*", will corrupt retentive memory, because the value of the retain variables that were saved does not match the retrieved application variables in the restored memory.

When the Non-Active PLC realizes that has a different project from the Active PLC, it executes the following actions:

- Negotiates automatic project synchronization with the Active PLC
- In case it's in the Stand-by or Starting state, it switches to the Not-Configured state and remains in it until the projects are synchronized again. After that, returns automatically to the Stand-by state
- In case it's in the Not-Configured or Inactive state, the STAND-BY button from the PX2612 panel must be pressed or an equivalent command must be executed. This way, after the project synchronization, it goes out from the Not-Configured state and can go to Stand-by state, or go back to the Inactive state if there's a failure

6.5.6. Offline Download of Modifications with Process Control Interruption

In the present section, it's defined a procedure to execute an offline download which interrupts the process control. Such situation is acceptable in specific process types and during programmed process stopping.

An offline download from this type must be executed connecting MasterTool to the NET 1 channel of the Active PLC using the exclusive IP address from the Active PLC (IP Address PLC X). Before beginning an offline download in the Active PLC the user receives two MasterTool warnings:

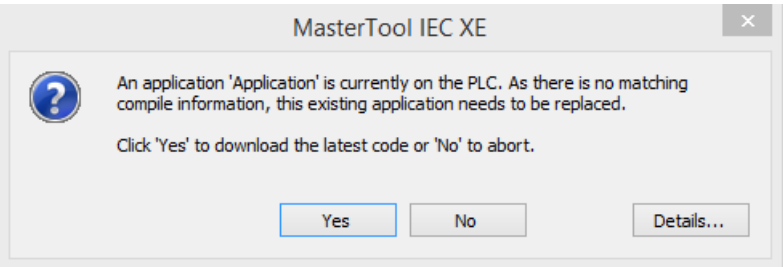


Figure 183: Offline Download Warning

By pressing *Yes*, the project is downloaded. When an offline download is performed, the process' control is interrupted, because the project is sent to the Active PLC, which will leave the Run state, and therefore will be in the Not-Configured state.

Another important point is that if the other PLC is in Stand-by state, it must be switched to Inactive state, e.g. pressing the PX2612 INACTIVE button on this PLC. This way, the turn off of this PLC by the other PLC and its take over as Active is avoided.

ATTENTION

When the Active PLC goes out from the Run mode and goes to Not-Configured, if the other PLC was forgotten in Stand-by state, it takes over as Active and switches off the PLC which has just gone from Active to Not-Configured. In this case, thus, the offline download can't be completed because the PLC connected to MasterTool is off.

When the offline download finishes, it's possible to restart the PLC program execution where the application was downloaded (put in Run again). After a few seconds, this PLC takes over again the Active state.

After this PLC takes the Active state again, the other PLC can go out from the Inactive state, e.g. pressing the PX2612 STAND-BY button on it. This causes the transition of this PLC to the Not-Configured state. This PLC remains in the Not-Configured state until the automatic project synchronization finishes. Then, it goes to Starting state and back to Stand-by state afterwards.

6.5.7. Previous Planning for Offline Modifications without Process Control Interruption

The following subsection – Previous Planning for Hot Modifications in Redundant PROFIBUS Networks – describes a very important procedure which allows the offline modifications download without interrupting the process control. Even though this procedure doesn't apply to any modification that demand offline download, it applies to the most used modifications.

However, in order to apply this procedure, the projects must be developed with a previous planning, especially for modification that affects the PROFIBUS network. The following subsections describe such previous planning for modifications that affect the PROFIBUS network and also other modifications.

6.5.7.1. Previous Planning for Hot Modifications in Redundant PROFIBUS Networks

Among the modifications that affect a PROFIBUS network and demand an offline download, the following are supported by the procedure which allows executing offline downloads without interrupt the process control, if the PROFIBUS network is redundant:

- Insert a new PROFIBUS network
- Insert a new Ponto Series remote
- Insert a new I/O module in a Ponto Series remote
- Modify parameters in Ponto Series remotes or in I/O modules in Ponto Series remotes

ATTENTION

It's possible to insert non-redundant remotes under a redundant PROFIBUS network, using the AL-2433 module (ProfiSwitch), as the example shown on Figure 159. However, such non-redundant remotes will suffer discontinuities (output deactivation) when the offline download is executed.

Next, the planning steps that must start at the creation of a new redundant PROFIBUS network are described.

6.5.7.1.1. Step 1 – Plan Future Expansion of the Remotes Inserted in the PROFIBUS Network Initial Version

At first, a list must be made of the I/O modules which compose each redundant PROFIBUS remote from the Ponto Series, in the PROFIBUS network initial version. The list must have the position where each I/O module is inserted in the remote rack.

Next, the future expansion of each remote must be planned. For that, a complementary list must be made, consisting in I/O modules which might be inserted in the future. In it, the position where each I/O module might be inserted in the remote rack must be listed.

ATTENTION

At the physical construction of these remotes (electric panels), it's strongly recommended to insert compatible bases with the future I/O modules in the respective positions. This way, when the I/O module insertion is necessary in this remote, there's no need for switching off the remote to insert the base. In case this detail isn't observed, it will be necessary to switch off the specific remote, as it's not possible a base hot insertion in the remote. It can be observed that the remote stopping in some cases can be tolerable, but not always.

ATTENTION

The original I/O module bases must be inserted in the first remote rack positions and the future I/O modules, in the last remote rack positions.

ATTENTION

It must be considered the limitations of the Ponto Series redundant remotes at constructing this list, as the PO5063V1 PROFIBUS Head and PO5063V5 PROFIBUS Redundant Head Utilization Manual, and PO5064 PROFIBUS Head and PO5065 PROFIBUS Redundant Head Utilization Manual. There are limits regarding the number of modules per remote, number of bytes per remote, current consuming per power supply, etc. These limits are verified automatically by the ProPonto. For further information, see the MT6000 MasterTool ProPonto Utilization Manual - MU299040.

6.5.7.1.2. Step 2 – Insert the Redundant PROFIBUS Network Initial Version in the Project

To insert the redundant PROFIBUS network initial version in the project, initially the two redundant NX5001 modules must be inserted in the rack, or use those already inserted by the redundancy wizard.

Next, each remote must be inserted in the device tree below these two NX5001, as well as the I/O modules under each remote.

Regarding the inserted I/O modules, there are two categories that must be treated differently:

- Those that are part of the PROFIBUS network initial version and will be installed immediately
- Those that will be used for future expansion

In the case of those that are part of the PROFIBUS network initial version, the module itself must be inserted in the device tree, in the planned remote correspondent position.

In the case of those that will be used for future expansion, a virtual module must be inserted in the planned correspondent position. A virtual module correspondent to a real module needs to allocate the same amount of I/O bytes than this real module. The virtual module insertion in the place of a real module avoids the real module absence diagnostics to be produced.

The following table shows real modules and its correspondent virtual modules:

Real Module	Correspondent Virtual Module
PO1000	PO9999 – 2 bytes Input
PO1001	PO9999 – 2 bytes Input
PO1002	PO9999 – 2 bytes Input
PO1003	PO9999 – 2 bytes Input
PO2020	PO9999 – 2 bytes Output
PO2022	PO9999 – 2 bytes Output

Table 204: Virtual Modules correspondent to the real modules

6.5.7.1.3. Step 3 – Allocate %I and %Q Variables Areas for the PROFIBUS Network considering Future Remote Expansion

As the NX5001, remotes and I/O modules were being inserted in the device tree in the previous step, %I and %Q variables were being allocated in three different areas:

- %I variables area for inputs
- %Q variables area for outputs
- %Q variables area for diagnostics

MasterTool executes the allocation of each one of these three variable areas in a continuous way, with no holes between them.

The initial and final address of each one of these three areas must be planned, considering the initially installed remotes in the network (see steps 1 and 2), but also remotes which might be inserted in the future in this same PROFIBUS network.

At defining the initial address of each area, it's important to reserve expansion for the device which allocates addresses immediately before the beginning of this area. On the other hand, at defining the final address of each area, it's important to reserve expansion for this PROFIBUS network.

Next, an example of such planning is shown, for %I variables area for inputs:

- PROFIBUS 1 network:
  - %IB0 ... %IB499 (addresses allocated to already installed remotes)
  - %IB500 ... %IB999 (addresses allocated to future remotes)
- PROFIBUS 2 network:
  - %IB1000 ... %IB1499 (addresses allocated to already installed remotes)
  - %IB1500 ... %IB1999 (addresses allocated to future remotes)
- Modbus TCP server:
  - %IB2000 ... %IB2999 (addresses allocated to current mapping)
  - %IB3000 ... %IB3999 (addresses allocated to future mapping)

For the two other areas (output %Q and diagnostic %Q) similar examples could be executed.

It's possible to predict the initially allocated and future expansion areas size using the following table which indicates the byte quantity allocated for the 3 areas for each module:

Module	Inputs %I (bytes)	Outputs %Q (bytes)	Diagnostic %Q (bytes)
NX5001	4	2	86
PO5063V5	0	0	25
PO5065	0	0	25
PO9100 (one each remote)	2	2	10
PO1000	2	0	10
PO2020	0	2	10
PO9999 – 2 bytes Output	0	2	10
PO9999 – 2 bytes Input	2	0	10

Table 205: %I and %Q variables allocation for PROFIBUS network modules

Note:

Variable Allocation: Further information regarding the size and type of memory allocated for each module can be found in the PROFIBUS-DP NX5001 Master Utilization Manual.

After executing the planning for the 3 areas (initial and final address of each area), the initial addresses must be inserted in the projected started in step 2.

At first, the parameter "*%Q Start Address of Module Diagnostics Area*" must be modified in the first NX5001 module, as shown on the table on the next figure. The planned initial address must be used for the diagnostic %Q variables.

Second, the first network I/O module must be found, starting with the NX5001, which allocate %I variables for inputs. At finding it, the correspondent "*Address*" parameter must be altered.

Third, the first network I/O module must be found, starting with the NX5001, which allocate %Q variables for outputs. At finding it, the correspondent "*Address*" parameter must be altered.

ATTENTION

At this moment it's recommended to verify the allocated address range for the 3 variable areas, verifying if the final addresses of each area are within the planned range, and if there's a good free area for expansion for new future remotes insertion.

6.5.7.2. Previous Planning for Other Hot Modifications

There are other hot modifications which, though they don't affect the PROFIBUS network, also demand offline downloading. Next, it's presented some examples of this type of modifications supported by the procedure which allow executing modifications offline download without interrupting the process control:

- NX5000 modules insertion (Ethernet)
- Ethernet or Serial communication I/O driver insertion
- Ethernet or Serial communication I/O driver new mapping insertion

On the other hand, the previous examples of modifications imply the direct representation %I and %Q variables allocation for diagnostics, inputs and outputs similar to discussed in step 3 from the previous planning for hot modifications which affect the PROFIBUS network (see Step 3 – Allocate %I and %Q Variables Areas for the PROFIBUS Network considering Future Remote Expansion).

This way, at inserting the NX5000 module, or an I/O Ethernet or Serial driver, the allocation of the 3 following areas must be planned for the inserted device:

- %I variables area for inputs
- %Q variables area for outputs
- %Q variables area for diagnostics

The Step 3 – Allocate %I and %Q Variables Areas for the PROFIBUS Network considering Future Remote Expansion section shows an example of group allocation of these areas, including PROFIBUS networks and an I/O driver (MODBUS TCP server).

6.5.7.3. Incompatibility of Applications

If the areas to be used in the future not be planned properly, the redundant memory areas may have to be altered, thus generating an incompatibility between the applications. This will result in only on PLC to remain in the Active state, with the other PLC remaining Inactive, without the possibility of synchronizing redundant data or application between the two PLCs.

This incompatibility is informed by the redundancy diagnostics at:
DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bApplicationIncompatible.

This diagnostic is active when the download of a new application is done to one of the PLCs, usually the Non-Active, with one of the following changes:

- Changes in the redundant memory areas, configured in the parameters of the NX4010 module
- Changes (create or remove) in the symbolic redundant variables, declared in redundant POUs or redundant GVLs

It is important to stress that, to make changes in symbolic redundant variables, the incompatibility problem will occur only if a new application download is done to one of the PLCs. In case that the modifications in symbolic redundant variables, and all the other modifications made in the project, fit into the group of Modifications which Allow Online Download, it's possible to do an Online Download of Modifications with no generation of incompatibility of applications between the PLCs.

6.5.7.4. Project Update due to MasterTool IEC XE Update

The MasterTool IEC XE programming tool is under a constant enhancement process, improving its features and adding new ones. When it is necessary to update the tool in a redundant system, the used project also needs to be updated. This update is done through the *Project/Project Update* menu, available in the tool. After updating the project, the Offline download without Process Control Interruption can be done.

6.5.7.4.1. Updating Project from Versions Previous to 2.00 to version 2.00 or Higher

Among the MasterTool IEC XE version changes there is a special case that must be planned more carefully to avoid stopping the process. The update of a project created with a version prior to 2.00 of the MasterTool IEC XE to version 2.00 or higher causes a reconfiguration in the area allocated for the Persistent Variables object. This reconfiguration was implemented aiming at optimizing the allocation of this area. However, if this object is present and marked as redundant in a project, this reconfiguration won't allow the data to be synchronized between the two project, always setting one of the Half-Cluster in Inactive State.

This way, if this situation happens, the NX3030 CPU software can detect it and stop the synchronization of the *Persistent Variables* object data until the two Half-Clusters' projects are the same, and, therefore, are using a project with the updated MasterTool IEC XE version. This situation won't stop the process, but if a correct update sequence is not followed, the data of the Persistent Variables object can be restarted.

In this case, the Offline Download sequence below must be followed:

- Change the Half-Cluster in Active state project, unmarking the *PersistentVars* object inside the *Redundancy Configuration* object. This download must be done as an Online change and for this to happen another change in the project must be done, e.g. declaring a new variable inside the NonSkippedPrg POU
- After the Online change, it's necessary to run the command *Create Boot Application* while online, with the PLC in Active state. This is necessary so that the application is synchronized with the Half-Cluster that passed to Not-Configured state after the download
- Update the project from version lower than 2.00 to version 2.00 or higher through the *Project/Project Update* menu in MasterTool IEC XE

- Disable the Project Synchronization through the *Online/Redundancy Configuration* menu
- Download the updated Project into the Half-Cluster that's in Stand-by state. A message will be displayed indicating the *PersistentVars* object memory area reorganization. The procedure must continue and by the end of the project download the Half-Cluster will remain in STOP with a redundancy state as Not-Configured
- Put the CPU in RUN. The Half-Cluster will change to Starting state and then to Stand-by. The Half-Cluster will synchronize its data with the one that's in Active state
- The data from the *PersistentVars* object must be copied from the Active Half-Cluster to the Stand-by manually or the receipt resource must be used
- Put the Active Half-Cluster to Stand-by. With this action, the other Half-Cluster will go to Active
- Enable the Project Synchronization through the *Online/Redundancy Configuration* menu. After this, the Half-Cluster in Stand-by state will go into Not-Configured state and will receive the project from the Half-Cluster in Active state. By the end of this process, the Half-Cluster state will go to Starting and then back to Stand-by
- Change the Project of the Half-Cluster in Active state marking the *PersistentVars* object inside the *Redundancy Configuration* object. This download must be done as an Online change, and for this to happen, another change in the project must be done, e.g. removing the variable declared at the beginning of this process
- After this, the Half-Cluster that was in Stand-by will pass to Not-Configured and will receive the Project from the Half-Cluster in Active state. By the end of this process the Half-Cluster state will change to Starting and then back to Stand-by

6.5.8. Exploring the Redundancy for Offline downloading of Modifications without Interruption of the Process control

In the section Offline and Online Modifications Download, it was informed that some modifications demand offline loading in the PLC where such modifications must be loaded. In these cases, the user has the option to interrupt the process control, according to the procedure defined in the section Offline Download of Modifications with Process Control Interruption. Therefore, it is usually necessary to program a process stop in advance, which is not always possible when an urgent modification is needed.

Thanks to the redundancy of the PLCs and, in some cases, thanks to the redundancy of the PROFIBUS network, it is possible to carry out offline loads without interrupting the process control for most of the usually necessary modifications. To achieve this goal, it is necessary to carefully follow a procedure, whose steps are described in the following subsections.

6.5.8.1. Step 1 – Verify Basic Requirements Attending

For the offline downloading with no interruption of the process control to be possible, the following basic requirements must be attended:

- The original project must have been created according to the recommendations of the Previous Planning for Offline Modifications without Process Control Interruption section
- The PLC must be redundant
- In case the modification affects the PROFIBUS network, it's necessary this network to be redundant. Such modifications may be:
  - New remotes insertion
  - I/O modules insertion in existent remotes, in previously reserved positions for correspondent virtual modules. For the remote not have to be switched off, there's the need of a base compatible with the new I/O module in the position reserved for it
  - Parameters modifications in remotes or existent I/O modules
- Both PLCs projects must be equalized and the Redundant Data Synchronization and Redundant Forcing List services must be working properly with no failure diagnostics. It can be stated these conditions are satisfied when there's a PLC in Active state and another in Stand-by state. In case the Non-Active PLC isn't in Stand-by state, the following diagnostics can be observed:
  - DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bRedDataSync = TRUE, indicates the success of the Redundant Data Synchronization service
  - DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bRedForceSync = TRUE, indicates the success of the Redundant Forcing List service
  - DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.dwApplicationCRC = DG_NX4010.tRedundancy.RedDgnRem .dwApplicationCRC, indicates both PLCs projects are equal

6.5.8.2. Step 2 – Don't Download in Group Modifications which can be downloaded Online

Modifications which can be downloaded online must not be downloaded together with modifications which must be downloaded offline without the process control interruption. When these two kinds of modifications are needed, they must always be loaded separately.

For the current procedure to be successful, it's absolutely necessary the modifications executed to don't cause any changes in the structure of the redundant variables exchanged between the Active and Non-Active PLC, through the Redundant Data Synchronization and Redundant Forcing List services. These two services must continue to working properly even while there are temporary differences between the PLCs. The modifications that must be loaded offline, and supported by this procedure do not affect the structure of redundant variables.

However, some modifications which can be loaded online can change the structure of redundant variables, e.g.:

- Insertion of symbolic variables (redundant or not) within a POU or GVL existing or in a new POU or GVL
- Removal of symbolic variables (redundant or not) within a POU or within existing GVL. The removal of a POU or GVL can also involve the removal of symbolic variables
- Change in size/structure of symbolic variables (redundant or not) in an existing POU or GVL

6.5.8.3. Step 3 – Previous Project Backup

Before editing the modifications that must be loaded offline without interrupting the process control, for safety reasons a backup of the project previous version must be run. It may be necessary to reinstall the previous version in case an error is committed during this procedure executing.

ATTENTION

The backup recommendation for all loaded versions in the PLCs may not be followed only in this specific procedure. It must be an usual practice.

6.5.8.4. Step 4 – Cares in Editing the Offline Downloaded Modifications

The offline downloaded modifications through this procedure, usually, are the following:

- Insertion of new devices in the devices tree
- Property or parameter change in devices existing in the devices tree

Such devices are normally the following:

- Modules such as PROFIBUS master (NX5001) or Ethernet modules (NX5000)
- Ponto Series PROFIBUS remotes
- I/O modules inside Ponto Series PROFIBUS remotes
- MODBUS communication I/O drivers
- Mapping of MODBUS communication drivers

The following cares must be taken at editing these project modifications:

- If a device existed in the previous project version, and continues existing in the modified version, the %I and %Q variables allocated for it must remain the same (command, diagnostic, inputs and outputs). Care must be taken for the inserted modifications don't change such allocations
- If a device was inserted in the modified project version, the %I and %Q variables allocated for it must not be allocated in the previous project version (command, diagnostic, inputs and outputs)

6.5.8.5. Step 5 – Inactive PLC Project Synchronism Disabling

In the procedures described in the Online Download of Modifications and Offline Download of Modifications with Process Control Interruption sections, the project is automatically synchronized from the Active PLC to the Non-Active PLC.

However, during the procedure of offline downloading without process control interruption, the project synchronism must be temporarily disabled. The Project synchronization disabling is explained in the section Project Synchronization Disabling and must be executed in the Non-Active PLC.

6.5.8.6. Step 6 – Physical Modifications Executing

At this moment, the physical modifications can be executed, such as:

- Install a new NX5000 module. This can be done through a module hot-insertion in each half-cluster rack, then connecting it to the Ethernet network
- Install a new redundant PROFIBUS network. The NX5001 can be hot-inserted in each half-cluster rack. Then, the redundant PROFIBUS network can be connected to them
- Install a new Ponto Series redundant remote. In this case, a remote head must be installed at a time, e.g. first in the network B and then in the network A:
  - To install the head in the network B, it may be necessary to open the cable or the contacts, thus perturbing the communication with the other heads already installed in the network B. Before doing that, all the operating active heads must be placed in the network A and the operating reserve heads in the network B
  - To install the head in the network A, it may be necessary to open the cable or the contacts, thus perturbing the communication with the other heads already installed in the network A. Before doing that, all the operating active heads must be placed in the network B and the operating reserve heads in the network A
- Install an I/O module in a base previously reserved for it, in an existent remote

6.5.8.7. Step 7 – Download the Offline Modifications in the Non-Active PLC

At first, MasterTool must be connected to the Non-Active PLC (see MasterTool Connection with a NX3030 CPU from a Redundant PLC section).

Next, the offline modifications must be downloaded. At doing it, the Non-Active PLC application is automatically interrupted (goes out of the Run mode).

6.5.8.8. Step 8 – Set the Non-Active PLC Back to Run Mode to make go back to Stand-by State

The offline load being finished, the Non-Active PLC can go back to Run mode.

A few seconds later, the Non-Active PLC must assume the Stand-by state.

In case the PLC doesn't assume the Stand-by state, the following problems may have caused this effect:

- The modifications executed changed the redundant variables structure, which prevents the correct execution of the Redundant Data Synchronization service. This can be verified through DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag .bRedDataSync (0 = failure) diagnostics in the Non-Active PLC. In this case, the modifications must be undone, recovering the previous project backup and restarting this procedure
- Other problems may eventually prevent the transition to the Stand-by state, even though this is unexpected. In this case, the diagnostics and the redundancy log must be observed

In case the PLC has assumed the Stand-by state, it's recommendable to verify if the projects are different between the Active and the Non-Active PLC. This can be made comparing the diagnostics DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag .dwApplicationCRC and DG_NX4010.tRedundancy.RedDgnRem.dwApplicationCRC in the Non-Active PLC (the CRCs must be different).

In case both projects are equal in the PLCs, it's possible that the project synchronism disabling (step 5) has not being properly executed. This can be verified through the diagnostic DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag .bProjectSyncDisable, which must be true in the Non-Active PLC. If it isn't true, the procedure must be returned to step 5.

6.5.8.9. Step 9 – Execute Switchover between Active and Stand-by PLCs

A switchover between the PLCs must be executed, e.g. pressing the STAND-BY button on the Active PLC. The Stand-by PLC, which has a new project with the modifications, takes over as Active. The Active PLC, which has the old project, takes over as Stand-by.

6.5.8.10. Step 10 – Projects Synchronism Enabling in the Active PLC

In the step 5, the project synchronism was disabled in the Non-Active PLC. It can be observed this PLC is now in Active state.

In this step, the project synchronism must be enabled again in this PLC. The screen and methodology used for it were described in the section Project Synchronization Disabling. But this time we need to select the "*Enable*" option from the combo-box. MasterTool must be connected to the Active PLC (see MasterTool Connection with a NX3030 CPU from a Redundant PLC section).

After enabling the project synchronism in the Active PLC, the user must verify if this command was successful, verifying if DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bProjectSyncDisable = 0, in the Active PLC.

As soon as the project synchronism is enabled again, the following action sequence is expected:

- The Non-Active PLC (Stand-by state), which already knows the different between both projects; goes out from the Stand-by state and goes to the Not-Configured state
- The modified project (new) is copied from the Active PLC to the other, temporarily in Not-Configured state
- As soon as the projects are synchronized again, the Not-Configured PLC goes to Starting state and then it's supposed to go back to Stand-by state

6.5.8.11. Step 11 – Optional Reorganization of PLC and PROFIBUS Networks in Active State

At the end of the procedure, for standardization or organization reasons, the user may execute a switchover for the PLCA assumes as Active, and for all remotes PROFIBUS heads in Active state are in the network A.

6.6. Redundancy Maintenance

6.6.1. Modules Hot Swapping in a Redundant PLC

In case of failure in a module from one of the PLCs (PLCA and PLCB), the module hot swapping may be necessary, without interrupt the process control. For that, the following steps must be followed:

- Verify if the half-cluster which won't be modified is in Active or Stand-by state, allowing it to take the process control
- To put the half-cluster having its module changed in Inactive state, through the Redundancy Control Panel PX2612 or the Redundancy Commands
- Execute the necessary exchanges in the Inactive half-cluster, as indicated in the CPU Configuration - General Parameters - How to do the Hot Swap section
- To put the half-cluster back to Stand-by or Active state, according to necessity

6.6.2. MasterTool Warning Messages

When MasterTool is with a redundant project open, or when it's connected to a NX3030 CPU identified as PLCA or PLCB, some special warning messages may occur, as described in the following subsections.

6.6.2.1. Blocking of Redundant or Non-Redundant Project Download

MasterTool doesn't allow the download of a redundant project, unless the CPU is NX3030 and is identified as PLCA or PLCB (see Identification of an NX3030 CPU section).

On the other hand, MasterTool doesn't allow the download of a non-redundant project in a NX3030 CPU identified as PLCA or PLCB.

In case any of these illegal actions is tried, MasterTool warns with a correspondent message.

6.6.2.2. Warnings before Commands which may stop the Active PLC

Some commands, as the following, may stop a PLC:

- Offline load after Online / Login
- Debug / Stop
- Debug / New Breakpoint
- Online / Reset (Warm, Cold, Origin)

In case MasterTool is logged to the Active PLC, and one of these commands is tried, before sending it to the Active PLC, MasterTool sends the following message and waits for authorization:

"*If the other PLC is in Stand-by State, it will assume as Active and turn-off this PLC. If not, this won't happen, but the automated process will no longer be controlled.*"

6.6.2.3. Alert before Logging in to Non-Active CP

In normal circumstances, it isn't usual MasterTool to connect to the Non-Active PLC. This way, when there's a try to execute this type of command, MasterTool sends the following warning:

"*You are logging in to a Non-Active PLC, and this is not usual. Are you sure you want to execute this command?*"

On the other hand, there are circumstances (not so usual) in which it's necessary to login in the Non-Active PLC, and in these cases the user must authorize the login. Such circumstances may occur, e.g.:

- For initial configurations, as described in Initial Downloading of a Redundant Project section
- For downloading offline a different project in the Non-Active PLC, as described in the Exploring the Redundancy for Offline downloading of Modifications without Interruption of the Process control section
- For monitoring or forcing the non-redundant variables in the Non-Active PLC

6.6.3. Redundancy Diagnostics on NX3030 CPU Graphic Display

Many diagnostics related to redundancy can be observed on the NX3030 CPU display.

6.6.3.1. CP Redundancy Status

The PLC redundancy state, described in Redundant CPU States, is seen in the three initial characters on the main screen second line, as shown in the section Graphic Display. The display screen is presented on initialization and again a few seconds after the navigation is finished (without pressing the NX3030 CPU button).

6.6.3.2. Screens below the REDUNDANCY Menu

There's a menu called REDUNDANCY, where below it there are several screens. The description and access of these screens are available in the Configuration – CPU's Informative and Configuration Menu section.

6.6.4. Redundancy Diagnostics Structure

The NX4010 module diagnostics area is mapped over direct representation %Q variables, and defined symbolic through the AT directive, in the GVL Module_Diagnostics.

This section is divided in two parts:

- DG_NX4010.tGeneral: General diagnostics for NX4010 operation. They are described in the Redundancy Link Module Technical Characteristics – CE114900
- DG_NX4010.tRedundancy: Redundancy specific diagnostics which are described within the section. This item is divided in other 6 data structures:
  - RedDgnLoc: Has redundancy diagnostics of the local PLC (connected), e.g. the PLC redundancy state. This section is described in Redundancy Diagnostics
  - RedDgnRem: It's a copy from the other PLC RedDgnLoc, received through Synchronism channels NETA / NETB. This way the local PLC has access to the remote PLC diagnostics. This section is described in Redundancy Diagnostics
  - RedCmdLoc: Has redundancy commands generated in this PLC (local), for instance, through write commands from a SCADA system or generated in POUs in this PLC (ActivePrg or NonSkippedPrg). This section is described in Redundancy Commands
  - RedCmdRem: It's a copy from the other PLC (remote) RedCmdLoc, received through Synchronism channels NETA / NETB. This section is described in Redundancy Commands
  - RedUsrLoc: Used to allow the user to exchange information between PLCA and PLCB. This section is described in User Information Exchanged between PLCA and PLCB
  - RedUsrRem: Used to allow the user to exchange information between PLCA and PLCB. This section is described in User Information Exchanged between PLCA and PLCB

It is important to stress that the redundancy diagnostics structures are refreshed only when occurs, with success, a new data synchronization. Therefore, until a new synchronization doesn't occur, the diagnostics will remain with the last read value.

Furthermore, the diagnostics structures from the remote PLC are read only, that is, values written to these structures will be overwritten in the next synchronization. Thus, is not possible to use the RedCmdRem structure to execute a command in the remote PLC. Always use the structure RedCmdLoc to execute commands.

6.6.4.1. Redundancy Diagnostics

The Redundancy Diagnostics may have several uses, such as:

- They can be consulted in order to verify the existence of a problem that needs to be solved
- Every time there are variations on them, such variations are inserted as events in the Redundancy Event Log. Consulting the history sequence of such events, a switchover cause may be discovered, for instance
- They can be referenced in the user application (ActivePrg or NonSkippedPrg). E.g. the PLC state can be tested and in case it's not active, a MODBUS RTU serial master I/O driver can be disabled, in NonSkippedPrg

ATTENTION

The DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync (defined next) must be tested to verify if the data structure RedDgnRem was successfully read from the remote PLC in the last MainTask cycle. In case this diagnostic value is 0 (false), this means the data structure RedDgnRem wasn't successfully read from the remote PLC, thus the RedDgnRem values may be invalid or obsolete.

As RedDgnRem is a copy from the other PLC RedDgnLoc, it can be concluded the two structures have the same format. These are divided in other four substructures:

- sGeneral_Diag: Redundancy general diagnostics
- sNETA_Diag: NETA synchronism channel diagnostics
- sNETB_Diag: NETB synchronism channel diagnostics
- sNET_Stat: Common statistics for the synchronism channels NETA and NETB, for failure and success counting in the synchronization services
- sGeneral_DiagExt: Redundancy general diagnostics, extended part (continuing of sGeneral_Diag)

The "sGeneral_Diag" substructure has the following fields for redundancy general diagnostics:

Direct Variable		**AT variable DG_NX4010.tRedundancy. RedDgnLoc.sGeneral_Diag.***	**Description**
Variable	**Bit**		
%QB(n+4)	0	bConfigDone	TRUE – The configuration process, executed in the Not-Configured state, has finished.
			FALSE – The configuration process, executed in the Not-Configured state, hasn't finished yet or wasn't executed.
	1	bConfigError	TRUE – The configuration process, executed in the Not-Configured state, has finished with errors. It's a system error, normally not expected. Get in contact with ALTUS support to report it. Also inform the ConfigErrorCode diagnostic value for the ALTUS support.
			FALSE – The configuration process has finished successfully or wasn't executed.
	2	bTooManyRedAreas	TRUE – The number of redundant areas exceeded the maximum allowed. It's a system error, normally not expected. Get in contact with ALTUS support to report it.
			FALSE – The number of redundant areas is within the expected.
	3	bTemporaryBufferTooSmall	TRUE – Intermediate data structure with insufficient size. It's a system error, normally not expected. Get in contact with ALTUS support to report it.

Direct Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	**Bit**		
			FALSE – Intermediate data structure is within the expected.
	4	bExchangeSync	TRUE – The Diagnostic and Commands Exchange synchronization service was executed successfully in this MainTask cycle.
			FALSE – The RedDgnRem structure has obsolete or invalid values, as it wasn't read from the other PLC (remote) in this cycle.
	5	bRedDataSync	TRUE – The Redundant Data Synchronization service was executed successfully in this MainTask cycle.
			FALSE – The Redundant Data Synchronization service wasn't executed successfully in this MainTask cycle.
	6	bRedForceSync	TRUE – The Redundant Forcing List Synchronization service was executed successfully in this MainTask cycle.
			FALSE – The Redundant Forcing List Synchronization service wasn't executed successfully in this MainTask cycle.
	7	bApplicationIncompatible	TRUE – The application isn't compatible between the PLCs. Was done a new application download with one of the following changes: Changes in redundant memory area; Changes in symbolic redundant variables. Whereas this diagnostic be TRUE, one of the PLCs will stay in Inactive state until the same application be present in the two PLCs. This implies in reload the old application or download the new application to both PLCs. More information about how to proceed can be found in section Redundant CPU Program Downloading.
			FALSE – The application is compatible between the PLCs.
	0	Reserved	Reserved bit.
	1	bProjectSyncDisable	TRUE – The project application and project archive will not be synchronized between the PLCs. It's a copy from the non-volatile variable used to enabling or disabling the project synchronization, as described in the Project Synchronization Disabling section. The project synchronization is disabled in the local or remote PLC. This way, it's enough to execute the disabling command in one PLC for the project synchronization to be disabled. The enabling and disabling project synchronization commands are described in the Project Synchronization Disabling section.

Direct Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	**Bit**		
%QB(n+5)			FALSE – The project application and project archive will be synchronized between the PLCs
	2	bIncompatibleFirmware	TRUE – Firmware version is incompatible between this CPU and the remote one.
			FALSE – Firmware version is compatible between this CPU and the remote one.
	3	bApplicationProjectDiff	TRUE – The project application between this CPU and the remote one is different.
			FALSE – The project application between this CPU and the remote one is equal.
	4	bProjectArchiveDiff	TRUE – The project archive between this CPU and the remote one is different.
			FALSE – The project archive between this CPU and the remote one is equal.
	5	bOnlineChangeApply	TRUE – Some alteration was done online in the application and it hasn't been synchronized yet with the stand-by PLC.
			FALSE – There wasn't alterations online in the application or these have been synchronized already with the stand-by PLC.
	6	bFailedRED	TRUE – Failure in the NX4010 module. The NX3030 CPU can't communicate with this module through bus, or there's a failure in the NX4010 microprocessor.
			FALSE – The NX4010 module is working properly.
	7	bFailedPBUS1A	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 1 A network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.
			FALSE – There aren't failures in the PROFIBUS 1 A network.
	0	bFailedPBUS1B	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 1 B network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.

Direct Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	**Bit**		
			FALSE – There aren't failures in the PROFIBUS 1 B network.
	1	bFailureProfibus_1	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 1 network. In case the PROFIBUS 1 network is redundant, FailurePROFIBUS_1 results from an AND logic between FailedPBUS1A and FailedPBUS1B. In case the PROFIBUS 1 network isn't redundant, FailurePROFIBUS_1 is a copy from FailedPBUS1A.
			FALSE – There aren't failures in the PROFIBUS network.
%QB(n+6)	2	bFailedPBUS2A	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 2 A network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.
			FALSE – There aren't failures in the PROFIBUS 2 A network.
	3	bFailedPBUS2B	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 2 B network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.
			FALSE – There aren't failures in the PROFIBUS 2 B network.
	4	bFailureProfibus_2	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 2 network. In case the PROFIBUS 2 network is redundant, FailurePROFIBUS_2 results from an AND logic between FailedPBUS2A and FailedPBUS2B. In case the PROFIBUS 2 network isn't redundant, FailurePROFIBUS_2 is a copy from FailedPBUS2A.
			FALSE – There aren't failures in the PROFIBUS 2 network.

Direct Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	**Bit**		
	5	bProfibusVitalFailureAny	TRUE – This PLC can't communicate in the master state (active or passive) with at least one of the PROFIBUS networks configured in vital failure mode.
			FALSE – There aren't failures in the PROFIBUS networks configured in vital failure.
	6	bProfibusVitalFailureAll	TRUE – This PLC can't communicate in the master state (active or passive) with all the PROFIBUS networks configured in vital failure mode.
			FALSE – There aren't failures in the PROFIBUS networks configured in vital failure.
	7	bTurnOffOtherPLC_Normal	TRUE – This PLC is closing the PX2612 relay to keep the other PLC off in normal conditions and not due to PX2612 panel test.
			FALSE – The PX2612 relay is on (bTurnOffOtherPLC_TestMode) or off.
%QB(n+7)	0	bTurnOffOtherPLC_TestMode	TRUE – This PLC is closing the PX2612 relay to keep the other PLC off due to PX2612 panel test mode.
			FALSE – The PX2612 relay is on (bTurnOffOtherPLC_Normal) or off.
	1	bActiveLED	TRUE – The PX2612 LED ACTIVE is on.
			FALSE – The PX2612 LED ACTIVE is blinking (bBlinkActiveLED) or off.
	2	bBlinkActiveLED	TRUE – The PX2612 LED ACTIVE is blinking.
			FALSE – The PX2612 ACTIVE is on (bActiveLEDl) or off.
	3	bStandbyLED	TRUE – The PX2612 LED STAND-BY is on.
			FALSE – The PX2612 LED ACTIVE is blinking (bBlinkStandbyLED) or off.
	4	bBlinkStandbyLED	TRUE – The PX2612 LED STAND-BY is blinking.
			FALSE – The PX2612 LED STAND-BY is on (bStandbyLED) or off.
	5	bInactiveLED	TRUE – The PX2612 LED INACTIVE is on.
			FALSE – The PX2612 LED INACTIVE is off or blinking (bBlinkInactiveLED).
	6	bBlinkInactiveLED	TRUE – The PX2612 LED INACTIVE is blinking.
			FALSE – The PX2612 LED INACTIVE is on (bInactiveLED) or off.
	7	bRedPanelTestMode	TRUE – The PX2612 panel is in test mode.

Direct Variable		AT variable DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.*	Description
Variable	**Bit**		
			FALSE – The PX2612 panel is in normal mode.
%QB(n+8)	-	ePLC_ID	This diagnostics inform this PLC identification: - 0 = non-redundant - 2 = PLCA - 3 = PLCB It's a copy from the non-volatile variable used to identify the PLC, as described in the Identification of an NX3030 CPU section. In the Initial Downloading of a Redundant Project section MasterTool command used to write on this non-volatile variable is described.
%QB(n+9)	-	eRedState	Informs the redundancy state of this PLC: - Not-Configured = 0 - Starting = 2 - Stand-by = 3 - Active = 4 - Inactive = 5
%QB(n+10)	-	ePreviousRedState	Previous RedState value before the data transition.
%QW(n+11)	-	wRedStateDuration	Measures for how long (milliseconds) the current redundancy state has been assumed. This time stops incrementing when reaches 65535 ms.
%QW(n+13)	-	wConfigErrorCode	Error code discovered during the configuration process in the Not-Configured state. See ConfigError diagnostics described previously.
%QD(n+15)	-	dwApplicationCRC	32 bits application project CRC, used to detect differences between the application projects of the 2 PLCs.
%QD(n+19)	-	dwArchiveCRC	32 bits project archive CRC, used to detect differences between the project archive of the 2 PLCs.
%QD(n+23)	-	dwFirmwareVersion	This PLC firmware version, used to verify the compatibility between both PLC firmware.
%QD(n+27)	-	dwIECTimer	The IEC Timer synchronization is necessary for the bump-less operation of some function block as TON and TOF. Through this diagnostic the IEC Timer from the Active PLC is received and updated in the Non-Active PLC, since the Diagnostics and Commands Exchange service has been executed successfully. The counting starts at 0 and is incremented up to 4294967295. After counting overflow restarts with 0.

Direct Variable		AT variable DG_NX4010.tRedundancy. RedDgnLoc.sGeneral_Diag.*	Description
Variable	**Bit**		
%QW(n+31)	-	wCycleCounter	16 bits counter, used as sequence auxiliary information in the Redundancy Event Log. In the Active PLC, it's incremented each MainTask cycle. In the Non-Active PLC, receives a copy of the value existent in the Active PLC, since the Diagnostics and Commands Exchange service has been executed successfully. The counting starts at 0 and is incremented up to 65535. After counting overflow restarts with 0.

Table 206: Redundancy General Diagnostics

Notes:

Diagnostics Structures Visualization: The diagnostics structures added to the project can be visualized accessing the Library Manager from the tree view in the MasterTool IEC XE window. With that it's possible to visualize all data types defined in the structure.

Direct Representation Variables: The "n" represents the configured value in the NX4010 module, through MasterTool IEC XE software, as a %Q Start Address of Module Diagnostics. This definition is true for all diagnostics structure.

AT Directive: The AT directive is a word reserved in the programming software which is connected to a variable address. In case of a NX4010 module the DG_NX4010 variable is related to the module diagnostics initial address.

The "sNETA_Diag" substructure has the following fields for NETA synchronism channels diagnostics:

DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync: When this diagnostic variable is with value FALSE, is not possible to define the state of some other diagnostics, such as bIncompatibleFirmware, bApplicationProjectDiff and bProjectArchiveDiff. There will not represent the correct value because they depend on the correct functioning of the communication between the two CPUs, so that information can be correctly generated.

Direct Variable		AT Variable DG_NX4010.tRedundancy. RedDgnLoc.sNETA_Diag.*	Description
Variable	**Bit**		
%QB(n+33)	0	bGeneralFailure	TRUE – The synchronism channel has some type of failure. The 3 next diagnostics will indicate the specific failure.
			FALSE – The synchronism channel is working properly.
	1	bInternalFailure	TRUE – The detected failure has its cause within this PLC. Such failures are treated in a special way.
			FALSE – The NX4010 module is working properly.
	2	bLinkDownFailure	TRUE – The AL-2319A cable is disconnected from the NX4010 module or broken in one of the PLCs.
			FALSE – The AL-2319A cable is connected to the NX4010 module.
	3	bTimeoutFailure	TRUE – This failure is reported in case a synchronization service hasn't been finished successfully within a specific timeout and failures from the type bInternalFailure or bLinkDownFailure haven't been found to justify that.

Direct Variable		AT Variable DG_NX4010.tRedundancy. RedDgnLoc.sNETA_Diag.*	Description
Variable	**Bit**		
			FALSE – The NX4010 module is working properly.
	4 .. 7	bReserved[4..7]	Reserved.

Table 207: NETA Interface Specific Diagnostics

The "sNETB_Diag" substructure has the following fields for NETB synchronism channels diagnostics:

Direct Variable		AT Variable DG_NX4010.tRedundancy. RedDgnLoc.sNETB_Diag.*	Description
Variable	**Bit**		
%QB(n+34)	0	bGeneralFailure	TRUE – The synchronism channel has some type of failure. The 3 next diagnostics will indicate the specific failure.
			FALSE – The synchronism channel is working properly.
	1	bInternalFailure	TRUE – The detected failure has its cause within this PLC. Such failures are treated in a special way.
			FALSE – The NX4010 module is working properly.
	2	bLinkDownFailure	TRUE – The AL-2319B cable is disconnected from the NX4010 module or broken in one of the PLCs.
			FALSE – The AL-2319B cable is connected to the NX4010 module.
	3	bTimeoutFailure	TRUE – This failure is reported in case a synchronization service hasn't been finished successfully within a specific timeout and failures from the type bInternalFailure or bLinkDownFailure haven't been found to justify that.
			FALSE – The NX4010 module is working properly.
	4 .. 7	bReserved[4..7]	Reserved.

Table 208: NETB Interface Specific Diagnostics

The "*sNET_Stat*" substructure has service success and failure statistics. The local and remote PLCs statistics can be restarted through commands:

```
//Local PLC
DG_NX4010.tRedundancy.RedCmdLoc.bResetNETStatisticsLocal := TRUE;
//Remote PLC
DG_NX4010.tRedundancy.RedCmdLoc.bResetNETStatisticsRemote := TRUE;
```

The substructure has the following counters:

Direct Variable	AT Variable DG_NX4010.tRedundancy. RedDgnLoc.sNET_Stat.*	Description
%QW(n+35)	wSuccessExchDgCmdSync	Success counting of the Diagnostics and Commands service (0 to 65535)
%QW(n+37)	wFailedExchDgCmdSync	Failure counting of the Diagnostics and Commands service (0 to 65535)
%QW(n+39)	wSuccessRedDataSync	Success counting of the Redundant Data Synchronization service (0 to 65535)
%QW(n+41)	wFailedRedDataSync	Failure counting of the Redundant Data Synchronization service (0 to 65535)
%QW(n+43)	wSuccessRedForceSync	Success counting of the Redundant Forcing List Synchronization service (0 to 65535)
%QW(n+45)	wFailedRedForceSync	Failure counting of the Redundant Forcing List Synchronization service (0 to 65535)

Table 209: Interface Specific Diagnostics

Note:
Counters: All counters return to zero when its limit value is exceeded.
The substructure "*sGeneral_DiagExt*" owns the following fields to general redundancy diagnostics:

Direct Variable		AT Variable DG_NX4010.tRedundancy .RedDgnLoc.sGeneral_DiagExt.*	Description
Variable	**Bit**		
	0	bFailedPBUS3A	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 3 A network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.
			FALSE – There aren't failures in the PROFIBUS 3 A network.
	1	bFailedPBUS3B	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 3 B network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.

Direct Variable		AT Variable DG_NX4010.tRedundancy .RedDgn-Loc.sGeneral_DiagExt.*	Description
Variable	**Bit**		
			FALSE – There aren't failures in the PROFIBUS 3 B network.
	2	bFailureProfibus_3	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 3 network. In case the PROFIBUS 3 network is redundant, FailurePROFIBUS_3 results from an AND logic between FailedPBUS3A and FailedPBUS3B. In case the PROFIBUS 3 network isn't redundant, FailurePROFIBUS_3 is a copy from FailedPBUS3A.
			FALSE – There aren't failures in the PROFIBUS 3 network.
%QB(n+47)	3	bFailedPBUS4A	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 4 A network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.
			FALSE – There aren't failures in the PROFIBUS 4 A network.
	4	bFailedPBUS4B	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 4 B network. The master mode (communicating with slaves) is assumed by the Active PLC. The passive mode (communicating with the active master) is assumed by the Non-Active PLC. This failure can also be indicated in case the NX5001 module has a microprocessor failure, or in case it can't communicate with the NX3030 CPU via bus.
			FALSE – There aren't failures in the PROFIBUS 4 B network.
	5	bFailureProfibus_4	TRUE – This PLC can't communicate in the master state (active or passive) in the PROFIBUS 4 network. In case the PROFIBUS 4 network is redundant, FailurePROFIBUS_4 results from an AND logic between FailedPBUS4A and FailedPBUS4B. In case the PROFIBUS 4 network isn't redundant, FailurePROFIBUS_4 is a copy from FailedPBUS4A.

Direct Variable		AT Variable DG_NX4010.tRedundancy .RedDgn-Loc.sGeneral_DiagExt.*	Description
Variable	**Bit**		
			FALSE – There aren't failures in the PROFIBUS 4 network.
	6	bFailureCommBusAny	TRUE – There is a failure in some communication fieldbus. As an example, an Ethernet port with MODBUS protocol configured to vital failure.
			FALSE – No fieldbus has failures.
	7	bFailureCommBusAll	TRUE – There is a failure in all fieldbus communication. As example, NET 1 and NET 2 are configured to vital and both are in failure.
			FALSE – There is at least one communication fieldbus working without failure.
%QB(n+48)	-	byFailedCommBusCount	Communication fieldbus quantity, which are reporting failures.
%QB(n+49)	0	bRemCpuKeepAliveNet1	TRUE – The NX3030 CPU is receiving Keep Alive packets from the other half-cluster's CPU, through NET 1. The Keep Alive packets are only sent in projects that don't use the PX2612 panel and that haven't a PROFIBUS network.
			FALSE – Keep Alive packets aren't being received.
	1	bRemCpuKeepAliveNet2	TRUE – The NX3030 CPU is receiving Keep Alive packets from the other half-cluster's CPU, through NET 2.
			FALSE – Keep Alive packets aren't being received.
	2..7	(Occulted reserved bits)	Bits reserved to future use. They aren't shown at the symbolic structure (hidden).
%QB(n+50)	-	abyReservedBytes[1..5]	5 reserved bytes to future use. They aren't shown at the symbolic structure (hidden).

Table 210: General Redundancy Diagnostics, Extended Part

6.6.4.2. Redundancy Commands

The structure command fields RedCmdLoc and RedCmdRem have a suffix which can be Local or Remote. E.g. there are the command fields StandbyLocal and StandbyRemote that have equivalent effect to the PX2612 panel STAND-BY button.

A command with Local suffix generated in RedCmdLoc must be executed in the local PLC (local). On the other hand, a command with Remote suffix generated in RedCmdLoc will be executed in another PLC (remote). This works as the following:

- The remote PLC, each MainTask cycle, receives the RecCmdLoc copy from the local PLC through NETA / NETB, and this copy is called RedCmdRem in it
- The remote PLC only executes the RedCmdRem commands with the Remote suffix

Example 1: if the local PLC is in Active state, and it's desired to switch it to the Stand-by state, the DG_NX4010.tRedundancy .RedCmdLoc.bStandbyLocal bit can be turned on in it.

Example 2: if the remote PLC is in Active state, and it's desired to switch it to the Stand-by state, the DG_NX4010.tRedundancy .RedCmdLoc.bStandbyRemote bit can be turned on in the local PLC. This may be useful, for instance, if the communication of a SCADA system is temporarily unavailable with the remote PLC. In this case, the command is written by the SCADA in the local PLC that retransmits to the remote PLC through NETA / NETB.

ATTENTION

If the DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bExchangeSync diagnostic is indicating a Diagnostics and Commands Exchange service failure, a command with Remote suffix isn't allowed to be transmitted to the remote PLC, thus, won't be executed.

To trigger a command, the RedCmdLoc correspondent bit must be turned on. This can be done through a SCADA system, executing writing via MasterTool or even turning the bit on inside a POU as ActivePrg or NonSkippedPrg.

The user doesn't need to worry with the command bit deactivating, which is automatically done by the redundancy manager:

- In case of commands executed in the local PLC (RedCmdLoc + commands with Local suffix), the bit is turned off as soon as the command is seen and executed
- In case of commands executed in the remote PLC (RedCmdRem + commands with Remote suffix):
  - In the remote PLC, the command is executed when the redundancy manager sees an up-going edge in the command bit
  - In the local PLC where the command was generated, the bit is turned off automatically in the next MainTask cycle

ATTENTION

There are two command bits which normally must be turned off by the user: DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal and DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal. Further details regarding these commands are described ahead in this section. In case the user forgets to turn them off, there are automatic mechanisms which are supposed to do it instead.

It's important to stress that any command activating or deactivating are registered in the Redundancy Event Log, which is important for the history analysis, e.g. to determine a switchover cause.

Next, the RedCmdLoc and RedCmdRem structure fields are defined:

Direct Variable		AT variable DG_NX4010.tRedundancy.RedCmdLoc.*	Description
Variable	**Bit**		
	0	bButtonTurnOnLocal	TRUE – It's a processed copy from the TURN ON PLCX button written on the PX2612 panel. This bit is activated 1 second after the button pressing and deactivated immediately at its releasing. It's important to stress that this bit will be activated when the TURN ON button on the remote PLC is pressed, as this type of command is always sent by the remote PLC.
			FALSE – The button TURN ON PLCX isn't pressed.
	1	bButtonStandbyLocal	TRUE – It's a processed copy from the STAND-BY button written on the PX2612 panel. This bit is activated 1 second after the button pressing and deactivated immediately at its releasing.
			FALSE – The button STAND-BY isn't pressed.
	2	bButtonInactiveLocal	TRUE – It's a processed copy from the INACTIVE button written on the PX2612 panel. This bit is activated 1 second after the button pressing and deactivated immediately at its releasing.
			FALSE – The button INACTIVE isn't pressed.

Direct Variable		AT variable DG_NX4010.tRedundancy.RedCmdLoc.*	Description
Variable	**Bit**		
%QB(n+55)	3	bAutoConfigLocal	TRUE – This diagnostics inform an automatic configuration request, necessary to let the Not-Configured state in some situations.
			FALSE – Automatic configuration request disabled.
	4	bTurnOnLocal	TRUE – This command produces an equivalent action to the TURN ON PLCX button on the PX2612 in the local PLC.
			FALSE – The TURN ON PLCX button on the local PLC isn't pressed.
	5	bStandbyLocal	TRUE – This command produces an equivalent action to the STAND-BY button on the PX2612 in the local PLC.
			FALSE – The STAND-BY button on the local PLC isn't pressed.
	6	bInactiveLocal	TRUE – This command produces an equivalent action to the INACTIVE button on the PX2612 in the local PLC.
			FALSE – The INACTIVE button on the local PLC isn't pressed.
	7	bResetNETStatisticsLocal	TRUE – This command resets the NETA / NETB statistics (see substructure SNET_Stat in RedDgnLoc and RedDgnRem). Such statistics are failure and success counters in synchronization services.
			FALSE – The reset commands for the NETA / NETB statistics in the local PLC wasn't activated.
	0	bTestModeLocal	TRUE – This command puts the PX2612 panel in test mode, allowing its components to be tested (LEDs, relays and buttons), as explained in PX2612 Panel Test section. The PX2612 test mode is only accepted when this bit is on both PLCs. Therefore, for the PX2612 to be in test mode, the PLC verifies if RedCmdLoc.TestModeLocal and RedCmdRem.TestModeLocal are both on. The RedDgnLoc.RedPanelTestMode diagnostic is turned on to inform that the PX2612 is really in test mode. Normally the user must turn off the TestModeLocal bit on both PLCs as soon as the PX2612 tests are concluded, but in case he forgets to do that, the bit will be turned off automatically 15 minutes after being turned on.
			FALSE – The command which puts the PX2612 panel in test mode is deactivated.

Direct Variable		AT variable DG_NX4010.tRedundancy.RedCmdLoc.*	Description
Variable	**Bit**		
%QB(n+56)	1	bTestRelayLocal	TRUE – This command is used to test the PX2612 NO relay and, consequently, the external NC relay too, used to, eventually, turn off the other PLC. This command is only accepted while the PX2612 is in test mode, being automatically switched off and ignored if the PX2612 isn't in this mode. Normally, the user must turn off the TestRelayLocal bit as soon as the relay test is concluded, but if it's forgotten, the bit is turned off as soon as the test mode is finished. It's important to stress this command is only accepted in the Active PLC, to avoid the Non-Active PLC to switch it off.
			FALSE – The command used to test the PX2612 NO relay is deactivated.
	2	bStandbyRemote	TRUE – This command produces an equivalent action to the STAND-BY button on the PX2612 in the remote PLC.
			FALSE – The STAND-BY button on the remote PLC isn't pressed.
	3	bInactiveRemote	TRUE – This command produces an equivalent action to the INACTIVE button on the PX2612 in the remote PLC.
			FALSE – The INACTIVE button on the remote PLC isn't pressed.
	4	bResetNETStatisticsRemote	TRUE – This command produces an equivalent action to the ResetNETStatisticsLocal button on the PX2612 in the remote PLC.
			FALSE – The reset commands for the NETA / NETB statistics in the remote PLC wasn't activated.
	5..7	bReserved[5..7]	Reserved.

Table 211: Redundancy Commands

6.6.4.3. User Information Exchanged between PLCA and PLCB

The Diagnostics and Commands Exchange Synchronization service, in each MainTask cycle, exchange the following data structures between both PLCs, using the NETA / NETB synchronism channels:

- Redundancy Diagnostics (RedDgnLoc and RedDgnRem), already described in the Redundancy Diagnostics Structure section
- Redundancy Commands (RedCmdLoc and RedCmdRem), already described in the Redundancy Commands section
- User Information Exchanged between PLCA and PLCB (RedUsrLoc and RedUsrRem), which are described in this section

The RedUsrLoc and RedUsrRem structures are simply a 128 bytes array, which utilization can be freely defined by the user. They allow the user to transfer, each cycle, 128 bytes of information from PLCA to PLCB, and other 128 bytes from PLCB to PLCA.

RedUsrRem is a copy from the other PLC RedUsrLoc, received through NETA / NETB. A specific PLC writes information on RedUsrLoc, which are read in the RedUsrRem of the other PLC.

These data structures have many utilities. E.g. supposing the SCADA system is connected only to the Active PLC and it's desired to visualize some information from the Non-Active PLC. The Non-Active PLC can put this information in these data structures. Among such information, for instance, might be some not mapped diagnostics in RedDgnLoc and RedDgnRem.

6.6.4.4. Modbus Diagnostics used at Redundancy

To check for failure in all MODBUS Server configured in a MODBUS Client instance, there is a specific diagnosis in each MODBUS Client instance configured. The table below displays the diagnostics for this type of failure in a MODBUS Client instance called MODBUS_Symbol_Client.

Variable DG_MODBUS_Symbol_ Client.tDiag.*	**Description**
bAllDevicesCommFailure	TRUE – All servers configured at this Client shows error.
	FALSE – There is at least one Server configured in this Client that doesn't shows error.

Table 212: Modbus Client Diagnostic

When configured vital failure mode, this diagnostic is consulted and 3 seconds after it's state change from FALSE to TRUE, a switchover occurs if the other conditions for this event are satisfied.

6.6.4.5. Redundancy Event Log

MasterTool allows the observation of several logs for the Nexto PLC, among them the Redundancy Event Log. These messages, specific for redundancy, register in the System Log relevant modifications in the diagnostics data structure fields and redundancy commands structure data.

Each line presented in the log has the following columns:

- Time Stamp: event time and date, with resolution in milliseconds
- Severity: information, warning, error or exception
- Description: text that describes the event
- Component: component that has generated the event, and in the Redundancy Event Log case, is "*Redundancy Management*"

The "*Description*" column text has information about the event that happened.

An example of the *Description* column can be the following:

Redundancy new state (local): Starting

To access this screen, a double click must be done on the device (NX3030) in the device tree, and then the tab "*Log*" must be selected. There's a filter that allows selecting only the "*Redundancy Management*" component, to show only the redundancy events.

ATTENTION

Some diagnostics may point to possible failures during the redundant system initialization and in the tasks first cycles. But in a correct system function these diagnostics no longer indicate errors right after the system initialization.

6.6.5. PX2612 Panel Test

The PX2612 panel is composed by buttons, LEDs and relays. Many of these resources are not used very often, thus are rarely tested and the defects may not be detected. It's important to run tests from time to time in order to verify if these resources are working properly, to avoid obscure failures to prevent the PX2612 use when it's necessary.

6.6.5.1. Test Mode Entry

The first step to test the PX2612 is to set it to test mode. This is done turning on the DG_NX4010.tRedundancy.RedCmdLoc .bTestModeLocal command bit on both PLCs.

The PLC perceives that is in test mode when the following two bits are on:

- DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal (RedCmdLoc.bTestModeLocal on in this PLC)
- DG_NX4010.tRedundancy.RedCmdRem.bTestModeLocal (RedCmdLoc.bTestModeLocal on in the other PLC)

When both bits are on, the PLC turns on the DG_NX4010.tRedundancy.RedDgnLoc.sGeneral_Diag.bRedPanelTestMode diagnostic, to inform that the PX2612 is in test mode.

6.6.5.2. Test Mode Manual and Automatic Outputs

The user can finish the test mode manually, turning off the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal bit in both PLCs. Actually turning it off in just one PLC is enough, as the test mode demands this bit to be on in both PLCs. However, this practice is recommended.

In case the user forgets to turn off the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal bit, it's automatically turned off 15 minutes after being turned on, finishing automatically the test mode.

6.6.5.3. LEDs Testing

Thus, during the test mode, the 6 LEDs must blink, losing its normal utility, which is showing the redundancy state.

6.6.5.4. Buttons Test

At pressing a button in the test mode, a correspondent LED stops blinking, and remains on. The following table presents the connection between the pressed button and the LED which remains on.

Tested button	Correspondent LED
TURN ON PLC A	ACTIVE PLC B
STAND-BY PLC A	STAND-BY PLC A
INACTIVE PLC A	INACTIVE PLC A
TURN ON PLC B	ACTIVE PLC A
STAND-BY PLC B	STAND-BY PLC B
INACTIVE PLC B	INACTIVE PLC B

Table 213: Connection between buttons and LEDs in the PX2612 button test

It can be observed that normally the LED is on the pressed button side, except for the TURN ON PLCx.

Before the LED remains on, it's necessary to hold the button for, at least, 1 second. The LED returns to blinking after it's released.

During the test mode, the buttons don't allow the execution of functions which would be executed out of this mode, such as to cause a redundancy state change.

6.6.5.5. Relay Test

To test the relays, it was created the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal bit. At turning on this bit, if the PLC is in test mode and in Active state, it activates the relay, which must cause the other PLC (Non-Active) switching off. Turning off the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal, the relay is released, allowing the other PLC reactivating.

The command has no effect in the Non-Active PLC, to prevent it turns off the Active PLC.

The user must cause a switchover between PLCs (Active x Non-Active) in order to test the relay in both PLCs.

When the PLC which was switched off is reactivated and restarted, it returns with the DG_NX4010.tRedundancy.RedCmdLoc .bTestModeLocal off, thus the test mode is canceled. The DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal bit must be turned on again in this PLC and set it to Active state before testing its relay.

When the test mode is finished, the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit is automatically turned off, in case the user has forgotten it on.

6.6.5.6. Suggested Sequence for PX2612 Test Executing

The following sequence is suggested in order to execute the PX2612 tests:

- Turn on the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal command bit in both PLCs (PLCA and PLCB).
- It must be observed if the 6 LEDs are blinking.
- Press, one at a time, the 6 buttons and verify if the correspondent LED stops blinking and remain on while the button is pressed. It must be remembered the button must remain pressed for, at least, one second before the LED stops blinking and remains on, and that the LED returns to blinking after the button is released.
- Turn on the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit in the Active PLC. It must be observed the Non-Active PLC switching off.
- Turn off the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit in the Active PLC. It must be observed the Non-Active PLC switching on.
- Wait until the Non-Active PLC is restarted and assumes the Stand-by state. The test mode is active as the DG_NX4010 .tRedundancy.RedCmdLoc.bTestModeLocal bit is turned off at the restarting in Stand-by mode PLC.
- Cause a switchover between PLCs, pressing the Active PLC STAND-BY button. The normal use of the STAND-BY button is possible because the test mode isn't active.
- Turn on the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal command bit in the new Active PLC, which has just gotten out the Stand-by state. This way, the test mode is reactivated.
- Turn on the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit in the Active PLC. It must be observed the Non-Active PLC switching off.
- Turn off the DG_NX4010.tRedundancy.RedCmdLoc.bTestRelayLocal command bit in the Active PLC. It must be observed the Non-Active PLC reactivating.
- Turn off the DG_NX4010.tRedundancy.RedCmdLoc.bTestModeLocal command bit in the Active PLC, to finish the test mode. It's not necessary to do this in the Stand-by PLC, as it has just initialized, with the DG_NX4010.tRedundancy .RedCmdLoc.bTestModeLocal bit off.

7. Maintenance

One feature of the Nexto Series is the abnormality diagnostic generation, whether they are failures, errors or operation modes, allowing the operator to identify and solve problems which occurs in the system easily.

The Nexto CPUs permit many ways to visualize the diagnostics generated by the system, which are:

- One Touch Diag
- Diagnostics via LED
- Diagnostics via System Web Page
- Diagnostics via Variables
- Diagnostics via Function Blocks

The first one is an innovating feature of Nexto Series, which allows a fast access to the application abnormal conditions. The second is purely visual, generated through two LEDs placed on the panel (DG and WD) and also through the LEDs placed in the RJ45 connector (exclusive for Ethernet connection). The next feature is the graphic visualization in a WEB page of the rack and the respective configured modules, with the individual access allowed of the operation state and the active diagnostics. The diagnostics are also stored directly in the CPU variables, either direct representation (%Q) or attributed (AT variable), and can be used by the user application, for instance, being presented in a supervisory system. The last ones present specific conditions of the system functioning.

These diagnostics function is to point possible system installation or configuration problems, and communication network problems or deficiency.

7.1. Module Diagnostics

7.1.1. One Touch Diag

The One Touch Diag (OTD), or single touch diagnostics, is an exclusive feature the Nexto Series brings for the programmable controllers. With this new concept the user can verify the diagnostics of any module connected to the system straight on the CPU graphic display with a single touch on the module Diagnostic Switch. This is a powerful diagnostic tool which can be used offline (with no need of supervisory or programming software) making easier to find and solve quickly possible problems.

The diagnostics key is placed on the CPU upper part, in an easy access place and, besides giving active diagnostics, allows the access to the navigation menu, described in the Configuration – CPU's Informative and Configuration Menu section.

The figure below shows the CPU switch placement:

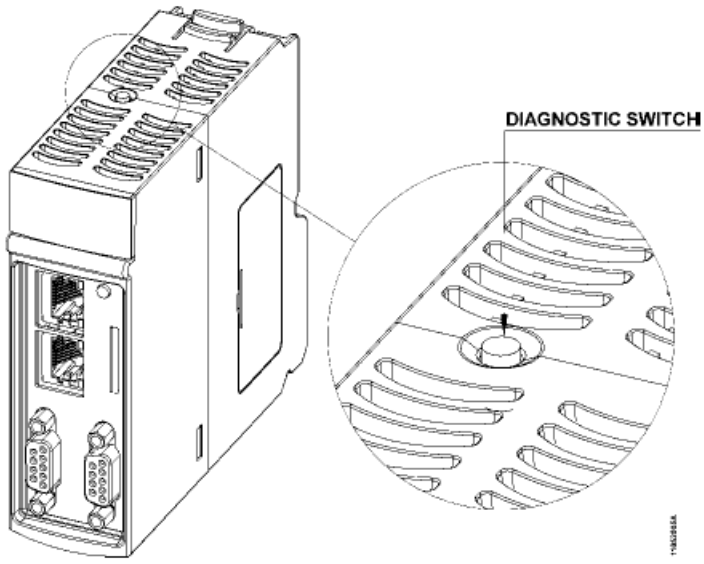


Figure 184: Diagnostic Switch

With only a short touch, the CPU starts to show the bus diagnostics (when active, otherwise shows the "NO DIAG" message). Initially, the Tag is visualized (configured in the module properties in the MasterTool IEC XE software, following

the IEC 61131-3 standard), in other words, the name attributed to the CPU, and after that all diagnostics are shown, through CPU display messages. This process is executed twice on the display. Everything occurs automatically as the user only has to execute the first short touch and the CPU is responsible to show the diagnostics. The diagnostics of other modules present on the bus are also shown on the CPU graphic display by a short press in the diagnostic module button, in the same presentation model of diagnostics.

The figure below shows the process starting with the short touch, with the conditions and the CPU times presented in smaller rectangles. It is important to stress the diagnostics may have more than one screen, in other words, the specified time in the block diagram below is valid for one of them.

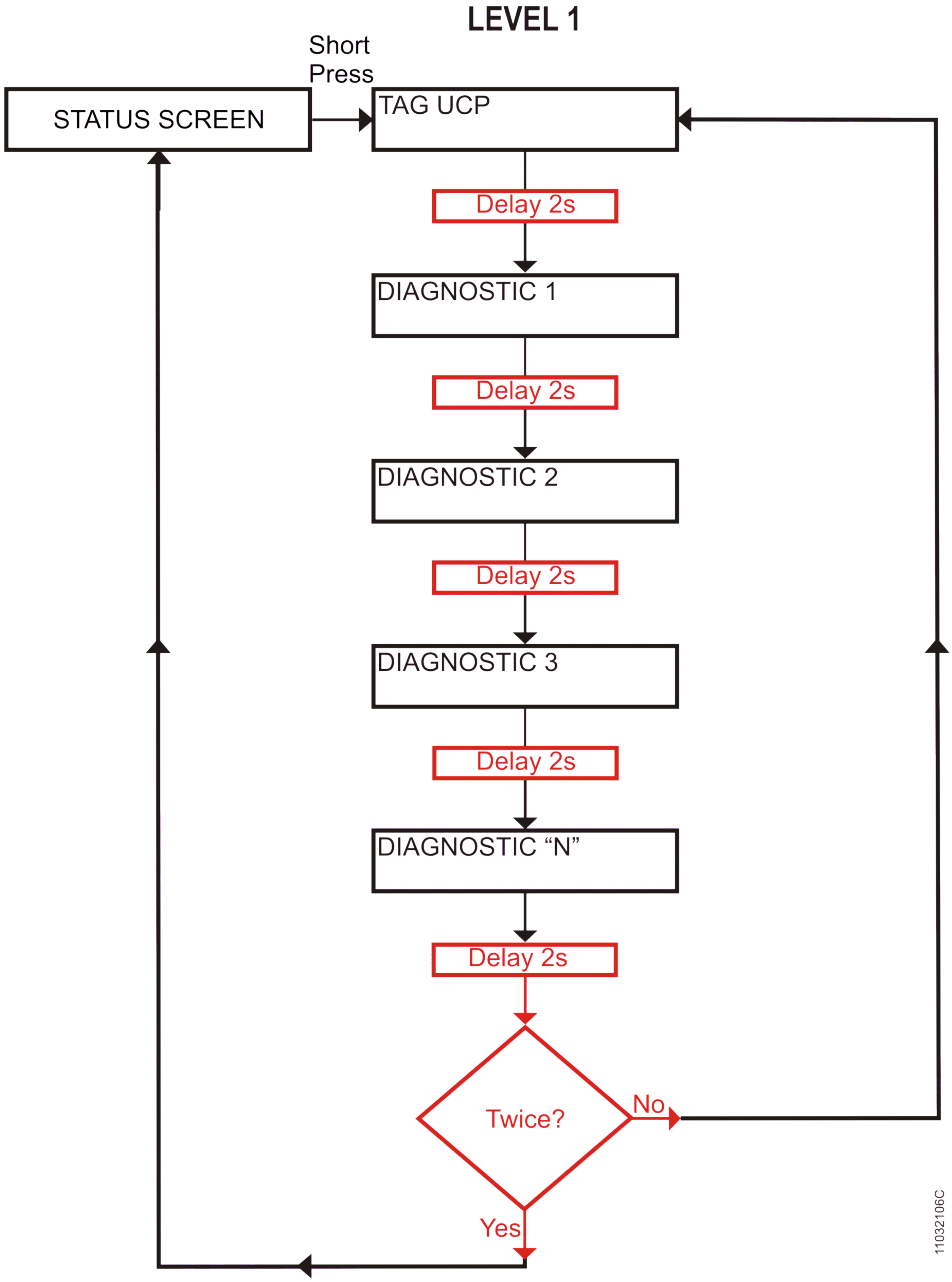


Figure 185: CPU Diagnostics Visualization

Before all visualization process be concluded, it is just to give a short touch on the diagnostic switch, at any moment, or press the diagnostic switch from any I/O module connected to the bus. Also, it is important to observe that the One Touch Diag could be available when the module could be in Operational Mode.

In case a long touch is executed, the CPU goes to navigation menu, which is described in the Configuration – CPU's Informative and Configuration Menu section.

The table below shows the difference between the short touch time, the long touch time and stuck button.

Touch type	Minimum time	Maximum time	Indication condition
No touch	-	59.99 ms	-
Short touch	60 ms	0.99 s	Release
Long touch	1 s	20 s	More than 1 s till 20 s
Locked Switch	20.01 s	(∞)	Diagnostics indication, see on Table 219

Table 214: One Touch Time

The messages presented on the Nexto CPU graphic display, correspondent to the diagnostics, are described in the Diagnostics via Variables section, on Table 219.

If any situation of stuck button occur in one of the I/O modules, the diagnostic button of this module will stop of indicate the diagnostics on CPU graphic display when is pressed. In this case, the CPU will indicate that there is a module with active diagnostics. To remove this diagnostic from the CPU, a hot swap must be done in the module where the diagnostic is active.

For further details on the procedure for viewing the diagnostics of the CPU or other bus modules, see description in the User Manual Nexto Series – MU214600.

7.1.2. Diagnostics via LED

This product have a LED for diagnostic indication (LED DG) and a LED for watchdog event indication (LED WD). The Tables 215 and 216 show the meaning of each state and its respective descriptions.

7.1.2.1. DG (Diagnostic)

Green	Red	Description	Causes	Priority
Off	**Off**	Not used	No power supply. Hardware problem	-
On	**Off**	All applications in execution mode (Run)	-	3 (Low)
Off	**On**	All applications in stopping mode (Stop)	-	3 (Low)
Blinking 2x	**Off**	Bus modules with diagnostic	At least, a bus module, including the CPU, is with an active diagnostic	1
Blinking 3x	**Off**	Data forcing	Some memory area is being forced by the user through MasterTool IEC XE	2
Off	**Blinking 4x**	Configuration or hardware error in the bus	The bus is damaged or is not properly configured	0 (High)

Table 215: Description of the Diagnostic LEDs States

7.1.2.2. WD (Watchdog)

Red LED	Description	Causes	Priority
Off	No watchdog indication	Normal operation	3 (Low)
Blinking 1x	Software watchdog	User application watchdog	2
On	Hardware watchdog	Damaged module and/or corrupted operational system	1 (High)

Table 216: Description of the Watchdog LED States

Notes:

Software Watchdog: In order to remove the watchdog indication, make an application reset or turn off and turn on the CPU again. This watchdog occurs when the user application execution time is higher than the configured watchdog time.

The diagnostics can be checked in the Exception.wExceptionCode variable, see on Table 223.

Hardware Watchdog: In order to reset any watchdog indication, as in the WD LED or in the Reset.bWatchdogReset operand, the module must be disconnected from the power supply.

In order to verify the application conditions in the module restart, see configurations on Table 44.

7.1.2.3. RJ45 Connector LEDs

Both LEDs placed in the RJ45 connectors, help the user in the installed physical network problem detection, indicating the network Link speed and the existence of interface communication traffic. The LEDs meaning is presented on table below.

Yellow	Green	Meaning
○	○	Network LINK absent
●	○	10 Mbytes/s network LINK
●	●	100 Mbytes/s network LINK
X	-	Ethernet network transmission or reception occurrence, for or to this IP address. Blinks on Nexto CPU demand and not every transmission or reception, in other words, it may blink on a lower frequency than the real transmission or reception frequency

Table 217: Ethernet LEDs Meaning

7.1.3. Diagnostics via System Web Page

Besides the previously presented features, the Nexto Series brings to the user an innovating access tool to the system diagnostics and operation states, through a System Web Page.

The utilization, besides being dynamic, is very intuitive and facilitates the user operations. The use of a supervisory system can be replaced when it is restricted to system status verification.

To access the desired CPU's System Web Page, it is just to use a standard browser (Internet Explorer 7 or superior, Mozilla Firefox 3.0 or superior and Google Chrome 8 or superior) and type, on the address bar, the CPU IP address (Ex.: http://192.168.1.1). First, the CPU information is presented, according to Figure 186:

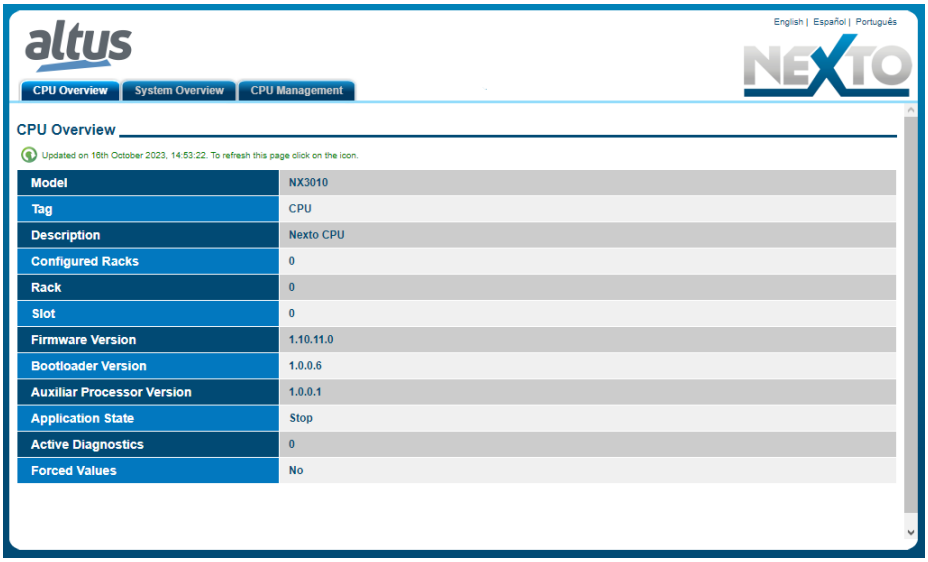


Figure 186: Initial Screen

There is also the *Bus Information* tab, which can be visualized through the Rack or the present module list (option on the screen right side). While there is no application on the CPU, this page will display a configuration with the largest available rack and a standard power supply, connected with the CPU. When the Rack visualization is used, the modules that have diagnostics blink and assume the red color, as shown on Figure 187. Otherwise a list with the system connected modules, Tags and active diagnostics number is presented:

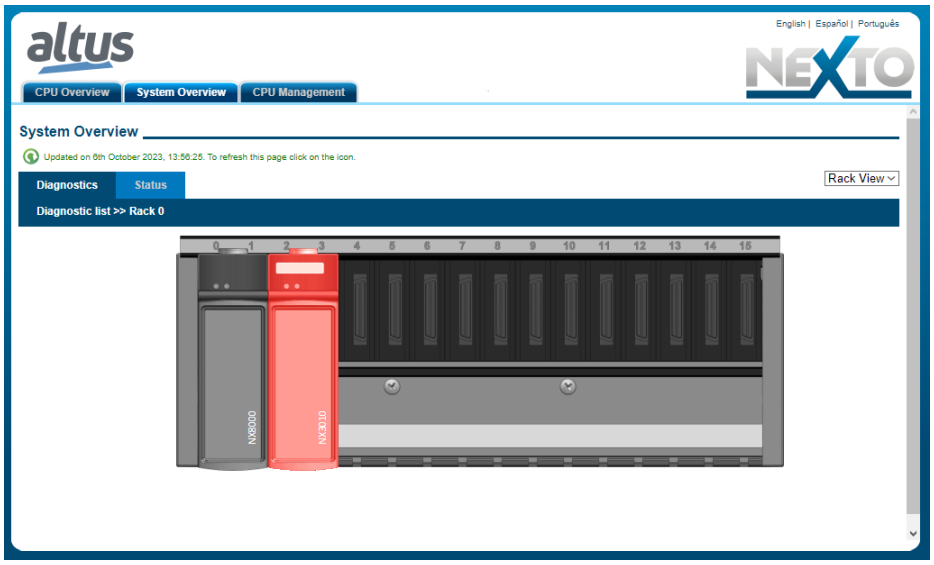


Figure 187: System Information

When the module with diagnostics is pressed, the module active(s) diagnostic(s) are shown, as illustrated on Figure 188:

ATTENTION

When a CPU is restarted and the application goes to exception in the system's startup, the diagnostics will not be valid. It is necessary to fix the problem which generates the application's exception so that the diagnostics are updated.

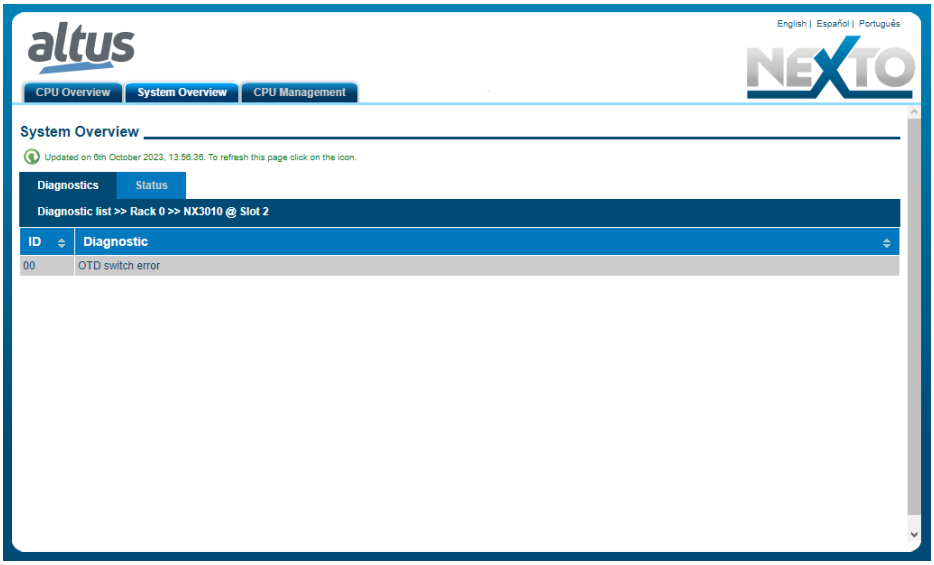


Figure 188: System Diagnostics

In case the Status tab is selected, the state of all detailed diagnostics is shown on the screen, as illustrated on Figure 189:

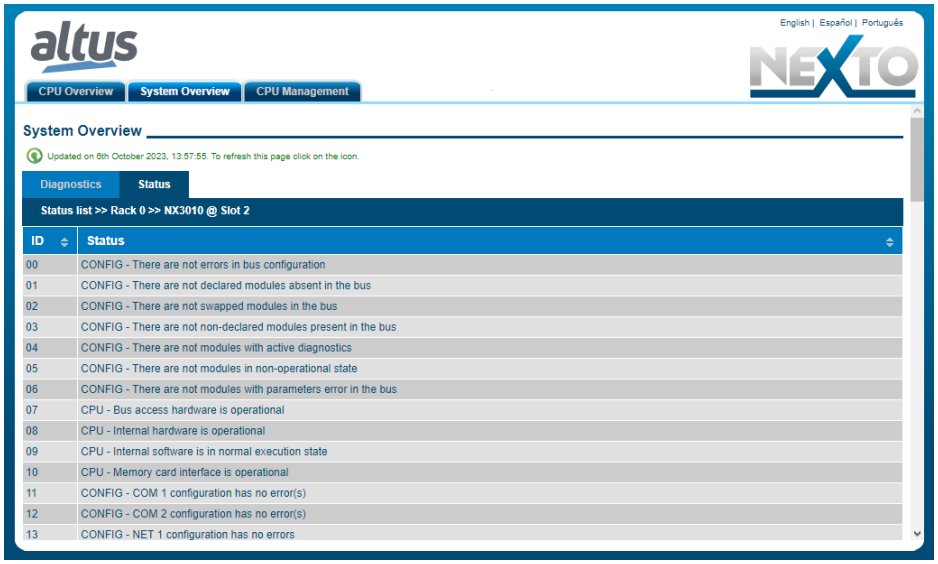


Figure 189: System Status

The user can choose to visualize two language options: Portuguese and English. Simply change in the upper right part of the screen to the desired language.

7.1.4. Diagnostics via Variables

The Nexto Series CPUs have many variables for diagnostic indication. There are data structures with the diagnostics of all modules declared on the bus, mapped on the variables of direct representation %Q, and defined symbolically through the AT directive, in the GVL System_Diagnostics created automatically by the MasterTool IEC XE.

The table below summarizes the diagnostic byte/words division:

Byte	Description
0 to 3	CPU summarized diagnostics.
4 to 693	CPU detailed diagnostics.

Table 218: CPU Diagnostics Division

7.1.4.1. Summarized Diagnostics

The table below shows the meaning of each CPU summarized diagnostic bit:

Direct Variable		Diagnostics Message	AT Variable DG_Module.tSummarized.*	Description
Variable	Bit			
-	-	NO DIAG	-	There is no active diagnostic.
%QB(n)	0	CONFIG. MISMATCH	bConfigMismatch	TRUE – There is a configuration problem in the bus, as the module inserted in the wrong position.
				FALSE – The bus is configured correctly.
	1	ABSENT MODULES	bAbsentModules	TRUE – One or more declared modules are absent.
				FALSE – All declared modules were detected in the bus.
	2	SWAPPED MODULES	bSwappedModules	TRUE – There are changed modules in the bus.
				FALSE – There are no changed modules in the bus.
	3	NON-DECLARED MODULES	bNonDeclaredModules	TRUE – One or more modules in the bus were not declared in the configuration.
				FALSE – All modules were declared.
	4	MODULES W/ DIAGNOSTICS	bModulesWithDiagnostic	TRUE – One or more modules in the bus are with active diagnostic.
				FALSE – There is no active diagnostic in the modules in the bus.
	5	MODULES W/ FATAL ERROR	bModuleFatalError	TRUE – One or more modules in the bus are in fatal error.
				FALSE – All modules are working properly.
	6	MODULES W/ PARAM. ERROR	bModuleParameterError	TRUE – One or more modules in the bus have parameterization error.
				FALSE – All modules are parameterized.
	7	BUS ERROR	bWHSBBusError	TRUE – Master indication there is failure in the WHSB bus.
				FALSE – The WHSB bus is working properly.
	0	HARDWARE FAILURE	bHardwareFailure	TRUE – CPU hardware failure.
				FALSE – The hardware is working properly.
	1	SOFTWARE EXCEPTION	bSoftwareException	TRUE – One or more exceptions generated by the software.

Direct Variable		Diagnostics Message	AT Variable DG_Module.tSummarized.*	Description
Variable	**Bit**			
%QB(n+1)				FALSE – No exceptions generated in the software.
	2	HARDWARE WATCHDOG	bHwWatchdogReset	TRUE - The CPU restarted by hardware watchdog at least once.
				FALSE - The CPU is operating normally.
	3	ERROR IN MEMORY CARD	bMemoryCardError	TRUE – The memory card is inserted in the CPU, but is not working properly.
				FALSE – The memory card is working properly.
	4	COM1 CONF. ERROR	bCOM1ConfigError	TRUE – Some error occurred during, or after, the COM 1 serial interface configuration.
				FALSE – The COM 1 serial interface configuration is correct.
	5	COM2 CONF. ERROR	bCOM2ConfigError	TRUE – Some error occurred during, or after, the COM 2 serial interface configuration.
				FALSE – The COM 2 serial interface configuration is correct.
	6	NET1 CONF. ERROR	bNET1ConfigError	TRUE – Some error occurred during, or after, the NET 1 Ethernet interface configuration.
				FALSE – The NET 1 Ethernet interface configuration is correct.
	7	NET2 CONF. ERROR	bNET2ConfigError	TRUE – Some error occurred during, or after, the NET 2 Ethernet interface configuration.
				FALSE – The NET 2 Ethernet interface configuration is correct.
%QB(n+2)	0	INVALID DATE/TIME	bInvalidDateTime	TRUE – The date or hour are invalid.
				FALSE – The date and hour are correct.
	1	RUNTIME RESET	bRTSReset	TRUE – The RTS (Runtime System) has been restarted at least once. This diagnostics is only cleared in the system restart.
				FALSE – The RTS (Runtime System) is operating normally.
	2	OTD SWITCH ERROR	bOTDSwitchError	TRUE – True in case the OTD key has been locked for more than 20 s at least once while the CPU was energized. This diagnostic is only cleared in the system restart.
				FALSE – The key is not currently locked or was locked while the CPU was energized.
	0	ABSENT RACK	bAbsentRacks	TRUE – One or more declared racks are absent.
				FALSE – There are no absent racks.
	1	DUPLICATED RACK	bDuplicatedRacks	TRUE – There are racks with a duplicated identification number.
				FALSE – There are no racks with a duplicated identification number.

Direct Variable		Diagnostics Message	AT Variable DG_Module.tSummarized.*	Description
Variable	**Bit**			
%QB(n+3)	2	INVALID RACK	bInvalidRacks	TRUE – There are racks with an invalid identification number.
				FALSE – There are no racks with an invalid identification number.
	3	NON DECLARED RACK	bNonDeclaredRacks	TRUE – There are racks with a non-declared identification number.
				FALSE – There are no racks with a non-declared identification number.
	4	DUPLICATED SLOT	bDuplicatedSlots	TRUE – There are some duplicated slot address.
				FALSE – There are no duplicated slot address.

Table 219: CPU Summarized Diagnostics

Notes:

Direct Representation Variable: "n" represents the value set in the CPU through the MasterTool IEC XE software, such as initial address diagnostics.

AT Directive: In the description of the symbolic variables which use the AT directive to make the mapping in direct addressing variables, the syntax that must be put before the desired summarized diagnostic is DG_Module.tSummarized, when the Module word is replaced by the used CPU. E.g. for the incompatible configuration diagnostic it must be used the variable: DG_NX3010.tSummarized.bConfigMismatch. The AT directive is a word reserved in the programming software, used only for diagnostic indication.

Configuration Mismatch: The incompatible configuration diagnostic is generated if one or more modules of the rack does not correspond to the declared one, so, in the absence or different modules conditions. The modules inserted in the bus that were not declared in the project are not considered.

Swapped Modules: If only two modules are changed between themselves in the bus, then changed diagnostic can be identified. Otherwise, the problem is treated as “Incompatible Configuration”.

Modules with Fatal Error: In case the modules with fatal error diagnostic is true, it must be verified which is the problematic module in the bus and send it to Altus Technical Assistance, as it has hardware failure.

Module with Parameterization Error: In case the parameterization error diagnostic is true, it must be verified the module in the bus are correctly configured and if the firmware and MasterTool IEC XE software version are correct. If the problem occurred when inserting a module on the bus, make sure the module supports hot swapping.

Bus Error: Considered a fatal error, interrupting the access to the modules in the bus. In case the bus error diagnostic is true, an abnormal situation due to the hot exchange configuration selected might have occurred or a hardware problem in the bus communication lines, then, contact Altus Technical Assistance.

Hardware Failure: In case the Hardware Failure diagnostic is true, the CPU must be sent to Altus Technical Assistance, as it has problems in the RTC, auxiliary processor, or other hardware resources.

Software Exception: In case the software exception diagnostic is true, the user must verify his application to guarantee it is not accessing the memory wrongly. If the problem remains, the Altus Technical Support sector must be consulted. The software exception codes are described next in the CPU detailed diagnostics table.

Diagnostic Message: The diagnostics messages can be visualized by the CPU graphic display using the OTD key or using the CPU’s System Web Page.

7.1.4.2. Detailed Diagnostics

The tables below contain Nexto Series' CPUs detailed diagnostics. It is important to have in mind the observations below before consulting them:

- **Visualization of the Diagnostics Structures:** The Diagnostics Structures added to the Project can be seen at the item "*Library Manager*" of MasterTool IEC XE tree view. There, it is possible to see all data types defined in the structure.
- **Counters:** All CPU diagnostics counters return to zero when their limit value is exceeded.
- **Direct representation variable:** "n" represents the value configured at the CPU through MasterTool IEC XE as the initial diagnostics address.
- **AT Directive:** At the description of symbolic variables that use the AT directive to map it in direct mapping variables, the syntax to be used before the desired summarized diagnostic is *DG_Module.tDetailed.*, where the word Module must be replaced by the CPU being used. The AT directive is a reserved word of the programmer, and some symbolic variables that use this directive indicate diagnostics.

Direct representation	Size	AT Variable DG_Modulo.tDetailed.*	Description
%QD(n+4)	DWORD	Target. dwCPUModel	NX3003 = 0x3003 NX3004 = 0x3004 NX3005 = 0x3005 NX3010 = 0x3010 NX3020 = 0x3020 NX3030 = 0x3030
%QB(n+8)	BYTE ARRAY(4)	Target. abyCPUVersion	Firmware version.
%QB(n+12)	BYTE ARRAY(4)	Target. abyBootloaderVersion	Bootloader version.
%QB(n+16)	BYTE ARRAY(4)	Target. abyAuxprocVersion	Auxiliary processor version.

Table 220: Target Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QX(n+20).0	BIT	Hardware. bAuxprocFailure	Failure in the communication between the auxiliary processor and the principal processor.
%QX(n+20).1	BIT	Hardware. bRTCFailure	The main processor is not enabled to communicate with the RTC (CPU's clock).
%QX(n+20).2	BIT	Hardware. bThermometerFailure [1]	Failure in the communication between the thermometer and the main processor.
%QX(n+20).3	BIT	Hardware. bLCDFailure	Failure in the communication between the LCD screen and the main processor.

Table 221: Hardware Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QW(n+21)	WORD	Exception. wExceptionCode	Exception code generated by the RTS. See Table 223.
%QB(n+23)	BYTE	Exception. byProcessorLoad	Level, in percentage (%), of charge in the processor.

Table 222: Exception Detailed Diagnostics Group Description

Note:

Exception Code: the code of the exception generated by the RTS (Runtime System) can be consulted below:

Code	Description	Code	Description
0x0000	There is no exception code.	**0x0051**	Access violation.
0x0010	Watchdog time of the IEC task expired (Software Watchdog).	**0x0052**	Privileged instruction.
0x0012	I/O configuration error.	**0x0053**	Page failure.
0x0013	Checksum error after the program download.	**0x0054**	Stack overflow.
0x0014	Fieldbus error.	**0x0055**	Invalid disposition.
0x0015	I/O updating error.	**0x0056**	Invalid maneuver.
0x0016	Cycle time (execution) exceeded.	**0x0057**	Protected page.
0x0017	Program online updating too long.	**0x0058**	Double failure.
0x0018	External references not resolved.	**0x0059**	Invalid OpCode.
0x0019	Download rejected.	**0x0100**	Data type misalignment.
0x001A	Project not loaded, as the retentive variables cannot be reallocated.	**0x0101**	Arrays limit exceeded.
0x001B	Project not loaded and deleted.	**0x0102**	Division by zero.
0x001C	Out of memory stack.	**0x0103**	Overflow.
0x001D	Retentive memory is corrupted and cannot be mapped.	**0x0104**	Cannot be continued.
0x001E	Project can be loaded but causes a drop later on.	**0x0105**	Watchdog in the processor load of all IEC task detected.
0x0021	Target of startup application does not match to the current target.	**0x0150**	FPU: Not specified error.
0x0022	Scheduled tasks error.	**0x0151**	FPU: Operand is not normal.
		0x0152	FPU: Division by zero.
0x0023	Downloaded file Checksum with error.	**0x0153**	FPU: Inexact result.
0x0024	Retentive identity is not correspondent to the current identity of the boot project program	**0x0154**	FPU: Invalid operation.
0x0025	IEC task configuration failure.	**0x0155**	FPU: Overflow.
0x0026	Application working with wrong target.	**0x0156**	FPU: Stack verification.
0x0050	Illegal instruction.	**0x0157**	FPU: Underflow.

Table 223: RTS Exception codes

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QB(n+25)	BYTE	RetainInfo. byCPUInitStatus	CPU Startup Status: 01: Hot start 02: Warm Start 03: Cold Start Note: These variables are reset in every powerup.
%QW(n+26)	WORD	RetainInfo. wCPUColdStartCounter	Increments when the CPU starts with loss of retentivity. (0 to 65535)
%QW(n+28)	WORD	RetainInfo. wCPUWarmStartCounter	Increments when the CPU starts normally with valid retain data. (0 to 65535)
%QW(n+30)	WORD	RetainInfo. wCPUHotStartCounter	Disturbance counter less than CPU power failure support time. (0 to 65535).

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QW(n+32)	WORD	RetainInfo. wRTSResetCounter	Counter of resets performed by RTS (Runtime System). (0 to 65535).

Table 224: RetainInfo Group Detailed Diagnostics

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QX(n+36).0	BIT	Reset. bBrownOutReset	The CPU was restarted due a failure in the power supply in the last startup.
%QX(n+36).1	BIT	Reset. bWatchdogReset	The CPU was restarted due the active watchdog in the last startup.

Table 225: Reset Detailed Diagnostics Group Description

Note:

Brownout Reset: The brownout reset diagnostic is only true when the power supply exceed the minimum limit required in its technical characteristics, remaining in low-voltage, i.e. without undergoing any interrupt. The CPU will identify the drop in supply and will indicate the power failure diagnostic. When the voltage is reestablished, the CPU will automatically reset and will indicate the brownout reset diagnostic.

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QX(n+37).0	BIT	Thermometer. bOverTemperatureAlarm [1]	Alarm generated due internal temperature at 85 °C or above it.
%QX(n+37).1	BIT	Thermometer. bUnderTemperatureAlarm [1]	Alarm generated due internal temperature at 0 °C or under it.
%QD(n+38)	DINT	Thermometer. diTemperature [1]	Temperature read in the internal sensor of the CPU.

Table 226: Thermometer Detailed Diagnostics Group Description

Note:

Temperature: In order to see the temperature directly in the memory address, a conversion must be made, since the data size is DINT and monitoring is done in 4 bytes. Therefore, it's recommended to use the associated symbolic variable, because it already provides the final temperature value.

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+42)	BYTE	Serial.COM1. byProtocol	Protocol selected in the COM 1: 00: Without protocol 01: MODBUS RTU Master 02: MODBUS RTU Slave 03: Other protocol
%QD(n+43)	DWORD	Serial.COM1. dwRXBytes	Counter of characters received from COM 1 (0 to 4294967295).
%QD(n+47)	DWORD	Serial.COM1. dwTXBytes	Counter of characters transmitted from COM 1 (0 to 4294967295).
%QW(n+51)	WORD	Serial.COM1. wRXPendingBytes	Number of characters left in the reading buffer in COM 1 (0 to 1024).
%QW(n+53)	WORD	Serial.COM1. wTXPendingBytes	Number of characters left in the transmission buffer in COM 1 (0 to 1024).
%QW(n+55)	WORD	Serial.COM1. wBreakErrorCounter	The transmitter is holding the data line at zero for too long, according to the databit configured.
%QW(n+57)	WORD	Serial.COM1. wParityErrorCounter	The received frame has the mismatched parity bit.

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QW(n+59)	WORD	Serial.COM1. wFrameErrorCounter	The received frame has the wrong start point, usually caused by a noise or baud rate mismatch.
%QW(n+61)	WORD	Serial.COM1. wRXOverrunCounter	When the receive ring buffer is full and starts to lose the old frames (too many frames not treated by the device).

Table 227: Serial COM 1 Detailed Diagnostics Group Description

Note:

Parity Error Counter: When the serial COM 1 is configured Without Parity, this error counter won't be incremented when it receives a message with a different parity. In this case, a frame error will be indicated.

Direct Variable	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+67)	BYTE	Serial.COM2. byProtocol	Protocol selected in the COM 2: 00: Without protocol 01: MODBUS RTU Master 02: MODBUS RTU Slave 03: Other protocol
%QD(n+68)	DWORD	Serial.COM2. dwRXBytes	Counter of characters received from COM 2 (0 to 4294967295).
%QD(n+72)	DWORD	Serial.COM2. dwTXBytes	Counter of characters transmitted through COM 2 (0 to 4294967295).
%QW(n+76)	WORD	Serial.COM2. wRXPendingBytes	Number of characters left in the reading buffer in COM 2 (0 to 1024).
%QW(n+78)	WORD	Serial.COM2. wTXPendingBytes	Number of characters left in the transmission buffer in COM 2 (0 to 1024).
%QW(n+80)	WORD	Serial.COM2. wBreakErrorCounter	The transmitter is holding the data line at zero for too long, according to the databit configured.
%QW(n+82)	WORD	Serial.COM2. wParityErrorCounter	The received frame has the mismatched parity bit.
%QW(n+84)	WORD	Serial.COM2. wFrameErrorCounter	The received frame has the wrong start point, usually caused by a noise or baud rate mismatch.
%QW(n+86)	WORD	Serial.COM2. wRXOverrunCounter	When the receive ring buffer is full and starts to lose the old frames (too many frames not treated by the device).

Table 228: Serial COM 2 Detailed Diagnostics Group Description

Note:

Parity Error Counter: When the serial COM 2 is configured Without Parity, this error counter won't be incremented when it receives a message with a different parity. In this case, a frame error will be indicated.

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QX(n+92).0	BIT	Ethernet.NET1. bLinkDown	Indicates link state on NET 1.
%QW(n+93)	WORD	Ethernet.NET1. wProtocol	Protocol selected in NET 1: 00: No protocol
%QX(n+93).0	BIT	Ethernet.NET1. wProtocol. bMODBUS_RTU_ETH_Client	MODBUS RTU client via TCP.
%QX(n+93).1	BIT	Ethernet.NET1. wProtocol. bMODBUS_ETH_Client	MODBUS TCP Client.

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QX(n+93).2	BIT	Ethernet.NET1. wProtocol. bMODBUS_RTU_ETH_Server	MODBUS RTU Server via TCP.
%QX(n+93).3	BIT	Ethernet.NET1. wProtocol. bMODBUS_ETH_Server	MODBUS TCP Server.
%QB(n+95)	STRING (15)	Ethernet.NET1. szIP	NET IP address 1.
%QB(n+111)	STRING (15)	Ethernet.NET1. szMask	NET 1 Subnet Mask.
%QB(n+127)	STRING (15)	Ethernet.NET1. szGateway	NET 1 Gateway Address.
%QB(n+143)	STRING (17)	Ethernet.NET1. szMAC	MAC NET 1 Address.
%QB(n+161)	BYTE ARRAY(4)	Ethernet.NET1. abyIP	NET IP address 1.
%QB(n+165)	BYTE ARRAY(4)	Ethernet.NET1. abyMask	NET 1 Subnet Mask.
%QB(n+169)	BYTE ARRAY(4)	Ethernet.NET1. abyGateway	NET 1 Gateway Address.
%QB(n+173)	BYTE ARRAY(6)	Ethernet.NET1. abyMAC	MAC NET 1 Address.
%QD(n+179)	DWORD	Ethernet.NET1. dwPacketsSent	Counter of packets sent via NET 1 port (0 to 4294967295).
%QD(n+183)	DWORD	Ethernet.NET1. dwPacketsReceived	Counter of packets received through NET 1 port (0 to 4294967295).
%QD(n+187)	DWORD	Ethernet.NET1. dwBytesSent	Count of bytes sent over NET 1 port (0 to 4294967295).
%QD(n+191)	DWORD	Ethernet.NET1. dwBytesReceived	Count of bytes received through NET 1 port (0 to 4294967295).
%QW(n+195)	WORD	Ethernet.NET1. wTXErrors	Counter of transmission errors via NET 1 port (0 to 65535).
%QW(n+197)	WORD	Ethernet.NET1. wTXFIFOErrors	Error counter in transmit buffer through NET 1 port (0 to 65535).
%QW(n+199)	WORD	Ethernet.NET1. wTXDropErrors	Connection loss counter when transmitting through the NET 1 port (0 to 65535).
%QW(n+201)	WORD	Ethernet.NET1. wTXCollisionErrors	Collision error counter when transmitting via NET 1 port (0 to 65535).
%QW(n+203)	WORD	Ethernet.NET1. wTXCarrierErrors	Transmission error counter on NET 1 port transmission (0 to 65535).
%QW(n+205)	WORD	Ethernet.NET1. wRXErrors	Counter of errors received via NET 1 port (0 to 65535).
%QW(n+207)	WORD	Ethernet.NET1. wRXFIFOErrors	Error counter in the receive buffer via NET 1 port (0 to 65535).
%QW(n+209)	WORD	Ethernet.NET1. wRXDropErrors	Connection loss counter when receiving via NET 1 port (0 to 65535).
%QW(n+211)	WORD	Ethernet.NET1. wRXFrameErrors	Frame error counter on reception via NET 1 port (0 to 65535).
%QW(n+213)	WORD	Ethernet.NET1. wMulticast	Counter of multicast packets through NET 1 port (0 to 65535).

Table 229: NET1 Ethernet Group Detailed Diagnostics

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QX(n+219).0	BIT	Ethernet.NET2. bLinkDown	Indicates link state in NET 2.

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QW(n+220)	WORD	Ethernet.NET2. wProtocol	Protocol selected in NET 2: 00: No protocol
%QX(n+220).0	BIT	Ethernet.NET2. wProtocol. bMODBUS_RTU_ETH_Client	MODBUS RTU client via TCP.
%QX(n+220).1	BIT	Ethernet.NET2. wProtocol. bMODBUS_ETH_Client	MODBUS TCP Client.
%QX(n+220).2	BIT	Ethernet.NET2. wProtocol. bMODBUS_RTU_ETH_Server	MODBUS RTU Server via TCP.
%QX(n+220).3	BIT	Ethernet.NET2. wProtocol. bMODBUS_ETH_Server	MODBUS TCP Server.
%QB(n+222)	STRING (15)	Ethernet.NET2. szIP	NET 2 IP address.
%QB(n+238)	STRING (15)	Ethernet.NET2. szMask	NET 2 Subnet Mask.
%QB(n+254)	STRING (15)	Ethernet.NET2. szGateway	NET 2 Gateway Address.
%QB(n+270)	STRING (17)	Ethernet.NET2. szMAC	MAC NET 2 address.
%QB(n+288)	BYTE ARRAY(4)	Ethernet.NET2. abyIP	NET 2 IP address.
%QB(n+292)	BYTE ARRAY(4)	Ethernet.NET2. abyMask	NET 2 Subnet Mask.
%QB(n+296)	BYTE ARRAY(4)	Ethernet.NET2. abyGateway	NET 2 Gateway Address.
%QB(n+300)	BYTE ARRAY(6)	Ethernet.NET2. abyMAC	MAC NET 2 address.
%QD(n+306)	DWORD	Ethernet.NET2. dwPacketsSent	Counter of packets sent via NET 2 port (0 to 4294967295).
%QD(n+310)	DWORD	Ethernet.NET2. dwPacketsReceived	Counter of packets received through NET 2 port (0 to 4294967295).
%QD(n+314)	DWORD	Ethernet.NET2. dwBytesSent	Count of bytes sent over NET 2 port (0 to 4294967295).
%QD(n+318)	DWORD	Ethernet.NET2. dwBytesReceived	Count of bytes received through NET 2 port (0 to 4294967295).
%QW(n+322)	WORD	Ethernet.NET2. wTXErrors	Counter of transmission errors via NET 2 port (0 to 65535).
%QW(n+324)	WORD	Ethernet.NET2. wTXFIFOErrors	Error counter in the transmission buffer through the NET 2 port (0 to 65535).
%QW(n+326)	WORD	Ethernet.NET2. wTXDropErrors	Connection loss counter when transmitting through the NET 2 port (0 to 65535).
%QW(n+328)	WORD	Ethernet.NET2. wTXCollisionErrors	Collision error counter when transmitting via NET 2 port (0 to 65535).
%QW(n+330)	WORD	Ethernet.NET2. wTXCarrierErrors	Transmission error counter on NET 2 port transmission (0 to 65535).
%QW(n+332)	WORD	Ethernet.NET2. wRXErrors	Counter of errors received via NET 2 port (0 to 65535).
%QW(n+334)	WORD	Ethernet.NET2. wRXFIFOErrors	Error counter in receive buffer via NET 2 port (0 to 65535).
%QW(n+336)	WORD	Ethernet.NET2. wRXDropErrors	Connection loss counter when receiving via NET 2 port (0 to 65535).
%QW(n+338)	WORD	Ethernet.NET2. wRXFrameErrors	Frame error counter on reception via NET 2 port (0 to 65535).
%QW(n+340)	WORD	Ethernet.NET2. wMulticast	Counter of multicast packets through NET 2 port (0 to 65535).

Table 230: NET2 Ethernet Group Detailed Diagnostics

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QB(n+346)	BYTE	UserFiles. byMounted	Indicates if the memory used to write user files is able to receive the data.
%QD(n+347)	DWORD	UserFiles. dwFreeSpacekB	User file memory free space in Kbytes.
%QD(n+351)	DWORD	UserFiles. dwTotalSizekB	User file memory storage capacity in Kbytes.

Table 231: Detailed Diagnostics Group UserFiles

Note:

User Partition: The user partition is a memory area reserved for data storage in the CPU. For example: files with PDF extension, files with DOC extension and other data.

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QB(n+356)	BYTE	UserLogs. byMounted	Status of the memory where user logs are inserted.
%QW(n+357)	WORD	UserLogs. wFreeSpacekB	User log memory free space in Kbytes.
%QW(n+359)	WORD	UserLogs. wTotalSizekB	User logs memory storage capacity in Kbytes.

Table 232: Detailed Diagnostics Group UserLogs

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+362)	BYTE	MemoryCard. byMounted	Status of the Memory Card: 00: Memory card not mounted 01: Memory card inserted and mounted
%QX(n+363).0	BIT	MemoryCard. bMemcardtoCPUEnabled	Protection level of the Memory Card: Data reading of the memory card by the authorized CPU.
%QX(n+363).1	BIT	MemoryCard. bCPUtoMemcardEnabled	Data writing in the memory card by the authorized CPU.
%QD(n+364)	DWORD	MemoryCard. dwFreeSpacekB	Free space in the Memory Card in Kbytes.
%QD(n+368)	DWORD	MemoryCard. dwTotalSizekB	Storage capacity of the Memory Card in Kbytes.

Table 233: MemoryCard Detailed Diagnostics Group Description

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+372)	BYTE	WHSB. byHotSwapAndStartupStatus	Informs the abnormal situation in the bus which caused the application stop for each mode of hot swapping. See Table 235 for more information.
%QB(n+373)	BYTE	WHSB. byReserved_0	Reserved.
%QD(n+374)	DWORD ARRAY (32)	WHSB. adwRackIOErrorStatus	Identification of errors in I/O modules, individually. For more information about this diagnostic, see the notes below.
%QD(n+502)	DWORD ARRAY (32)	WHSB. adwModulePresenceStatus	Status of presence of declared I/O modules in buses, individually. For more information about this diagnostic, see the notes below.
%QB(n+630)	BYTE	WHSB. byWHSBBusErrors	Counter of failures in the WHSB bus. This counter is restarted in the energization (0 to 255).

Table 234: WHSB Detailed Diagnostics Group Description

Notes:

Bus modules error diagnostic: Each DWORD from this diagnostic array represents a rack, whose positions are represented by the bits of these DWORDS. So, Bit-0 of the DWORD-0 is equivalent to position zero of the rack with address zero. Each one of these Bits is the result of an OR logic operation between the Incompatible Configuration (bConfigMismatch), absent modules (bAbsentModules), swapped modules (bSwappedModules), module with fatal error (bModuleFatalError) diagnostics and the operational state of the module in a certain position.

Module presence status: Each DWORD from this diagnostic array represents a rack, whose positions are represented by the bits of these DWORDS. So, Bit-0 from DWORD-0 is equivalent to position zero of the rack with address zero. So, if a module is present, this bit will be true. It's important to notice that this diagnostic is valid for all modules, except power supplies, CPUs and non-declared modules, e.g. those that are not in the rack on the respective position (bit remains in false).

Situations in which the Application Stops: The codes for the possible situations in which the application stops can be consulted below:

Code	Enumerable	Description
00	INITIALIZING	This state is presented while other states are not ready.
01	RESET_WATCHDOG	Application in Stop Mode due to hardware watchdog reset or runtime reset, when the option *"Start User Application After a Watchdog Reset"* is unmarked.
02	ABSENT_MODULES_HOT_SWAP_ DISABLED	Application in Stop Mode due to Absent Modules diagnostic being set when the Hot Swap Mode is *"Disabled"* or *"Disabled, for declared modules only"*.
03	CFG_MISMATCH_HOT_SWAP_ DISABLED	Application in Stop Mode due to Configuration Mismatch diagnostic being set when the Hot Swap Mode is *"Disabled"* or *"Disabled, for declared modules only"*.
04	ABSENT_MODULES_HOT_SWAP_ STARTUP_CONSISTENCY	Application in Stop Mode due to Absent Modules diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"* or *"Enabled, with startup consistency for declared modules only"*.
05	CFG_MISMATCH_HOT_SWAP_ STARTUP_CONSISTENCY	Application in Stop Mode due to Incompatible Configuration diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"* or *"Enabled, with startup consistency for declared modules only"*.
06	APPL_STOP_ALLOWED_TO_RUN	Application in Stop Mode and all consistencies executed successfully. The application can be set to Run Mode.

Code	Enumerable	Description
07	APPL_STOP_MODULES_NOT_READY	Application in Stop Mode and all consistencies executed successfully, but the I/O modules are not able to start the system. It is not possible to set the application to Run Mode.
08	APPL_STOP_MODULES_GETTING_READY_TO_RUN	Application in Stop Mode and all consistencies executed successfully. The I/O modules are being prepared to start the system. It is not possible to set the application to Run Mode.
09	NORMAL_OPERATING_STATE	Application in Run Mode.
10	MODULE_CONSISTENCY_OK	Internal usage.
11	APPL_STOP_DUE_TO_EXCEPTION	Application in Stop Mode due to an exception in the CPU.
12	DUPLICATED_SLOT_HOT_SWAP_DISABLED	Application in Stop Mode due to Duplicated Slots diagnostic being set when the Hot Swap Mode is *"Disabled"* or *"Disabled, for declared modules only"*.
13	DUPLICATED_SLOT_HOT_SWAP_STARTUP_CONSISTENCY	Application in Stop Mode due to Duplicated Slots diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"* or *"Enabled, with startup consistency for declared modules only"*.
14	DUPLICATED_SLOT_HOT_SWAP_ENABLED	Application in Stop Mode due to Duplicated Slots diagnostic being set when the Hot Swap Mode is *"Enabled, without startup consistency"*.
15	NON_DECLARED_MODULE_HOT_SWAP_STARTUP_CONSISTENCY	Application in Stop Mode due to Non Declared Modules diagnostic being set when the Hot Swap Mode is *"Enabled, with startup consistency"*.
16	NON_DECLARED_MODULE_HOT_SWAP_DISABLED	Application in Stop Mode due to Non Declared Modules diagnostic being set when the Hot Swap Mode is *"Disabled"*.

Table 235: Codes of the Situations in which the Application Stops

Direct representation	Size	AT Variable DG_Module.tDetailed.*	Description
%QB(n+631)	BYTE	Application. byCPUState	Informs the operation state of the CPU: 01: All user applications are in Run Mode 03: All user applications is in Stop Mode
%QX(n+632).0	BIT	Application. bForcedIOs	There is one or more forced I/O points.
%QX(n+632).1	BIT	Application. bNetDefinedByWeb	The NX30x0 CPU does not support IP exchange via the System Web Page. Therefore, the diagnosis will always remain FALSE.

Table 236: Application Detailed Diagnostics Group Description

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QX(n+633).0	BIT	SNTP. bServiceEnabled	SNTP service enabled.
%QB(n+634)	BYTE	SNTP. byActiveTimeServer	Indicates which server is active: 00: No active server. 01: Primary server active. 02: Secondary server active.

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QW(n+635)	WORD	SNTP. wPrimaryServerDownCount	Count of times the primary server was unavailable (0 to 65535).
%QW(n+637)	WORD	SNTP. wSecondaryServerDownCount	Count of times the secondary server was unavailable (0 to 65535).
%QD(n+639)	DWORD	SNTP. dwRTCTimeUpdatedCount	Count of times the RTC was updated by the SNTP service (0 to 4294967295).
%QB(n+643)	BYTE	SNTP. byLastUpdateSuccessful	Indicates status of last update: 00: Not updated. 01: Last update failed. 02: Last update was successful.
%QB(n+644)	BYTE	SNTP. byLastUpdateTimeServer	Indicates which server was used in the last update: 00: No updates. 01: Primary server. 02: Secondary server.
%QB(n+645)	BYTE	SNTP.sLastUpdateTime. byDayOfMonth	Day of last RTC update.
%QB(n+646)	BYTE	SNTP.sLastUpdateTime. byMonth	Month of last RTC update.
%QW(n+647)	WORD	SNTP.sLastUpdateTime. wYear	Year of last update of RTC.
%QB(n+649)	BYTE	SNTP.sLastUpdateTime. byHours	Time of last RTC update.
%QB(n+650)	BYTE	SNTP.sLastUpdateTime. byMinutes	Minute of last RTC update.
%QB(n+651)	BYTE	SNTP.sLastUpdateTime. bReservedAlign	Reserved for alignment.
%QB(n+652)	BYTE	SNTP.sLastUpdateTime. bySeconds	According to last RTC update.
%QW(n+653)	WORD	SNTP.sLastUpdateTime. wMilliseconds	Millisecond of last RTC update.

Table 237: SNTP Group Detailed Diagnostics

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QX(n+659).0	BIT	SOE[1]. bConnectionStatus	Client Connection Status 01
%QX(n+659).1	BIT	SOE[1]. bOverflowStatus	Client event queue status 01: FALSE - No overflow TRUE - Queue limit exceeded
%QB(n+660)	BYTE	SOE[1]. byReserved_0	Reserved
%QW(n+661)	WORD	SOE[1]. wEventsCounter	Customer Queue Event Counter 01
%QX(n+663).0	BIT	SOE[2]. bConnectionStatus	Client Connection Status 02
%QX(n+663).1	BIT	SOE[2]. bOverflowStatus	Client event queue status 02: FALSE - No overflow TRUE - Queue limit exceeded
%QB(n+664)	BYTE	SOE[2]. byReserved_0	Reserved.
%QW(n+665)	WORD	SOE[2]. wEventsCounter	Client queue event counter 02.

Table 238: SOE Group Detailed Diagnostics

Notes:

Synchronization of SOE group diagnostics in a system operating with Half-Cluster redundancy: When a project is configured with Half-Cluster redundancy, the SOE group diagnostics are not synchronized between the two Half-Clusters.

Updating SOE group diagnostics on transition to active state: When a Half-Cluster goes from Standby state to Active state, SOE group diagnostics are updated from the third cycle on.

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QD(n+667)	DWORD	Rack. dwAbsentRacks	Each bit represents an identification number of a rack, if any bit is TRUE, it means that the rack, with that identification number, is absent.
%QD(n+671)	DWORD	Rack. dwDuplicatedRacks	Each bit represents an identification number of a rack, if any bit is TRUE, it means that more than one rack is configured with the same identification number.
%QD(n+675)	DWORD	Rack. dwNonDeclaredRacks	Each bit represents a rack identification number, if any bit is TRUE, it means that there is a rack configured with an identification number that is not declared in the project.

Table 239: Rack Group Detailed Diagnostics

Direct representation	Size	AT DG variable_Modulo.tDetailed.*	Description
%QD(n+681)	DWORD	ApplicationInfo. dwApplicationCRC	32-bit CRC of the application. When the application is modified and sent to the CPU, a new CRC is calculated.

Table 240: ApplicationInfo Group Detailed Diagnostics

7.1.5. Diagnostics via Function Blocks

The function blocks allow the visualization of some parameters which cannot be accessed otherwise. The function regarding advanced diagnostics is in the *NextoStandard* library and is described below.

7.1.5.1. GetTaskInfo

This function returns the task information of a specific application.

Figure 190: GetTaskInfo Function

Below, the parameters that must be sent to the function for it to return the application information are described.

Input parameter	Type	Description
psAppName	POINTER TO STRING	Application name.
psTaskName	POINTER TO STRING	Task name.
pstTaskInfo	POINTER TO stTaskInfo	Pointer to receive the application information.

Table 241: GetTaskInfo Input Parameters

The data returned by the function, through the pointer informed in the input parameters are described on table below.

Returned Parameters	Size	Description
dwCurScanTime	DWORD	Task cycle time (execution) with 1 μs resolution.
dwMinScanTime	DWORD	Task cycle minimum time with 1 μs resolution.
dwMaxScanTime	DWORD	Task cycle maximum time 1 μs resolution.
dwAvgScanTime	DWORD	Task cycle average time with 1 μs resolution.
dwLimitMaxScan	DWORD	Task cycle maximum time before watchdog occurrence.
dwIECCycleCount	DWORD	IEC cycle counter.

Table 242: GetTaskInfo Output Parameters

Possible ERRORCODE:

- NoError: success execution;
- TaskNotPresent: the desired task does not exist.

Example of utilization in ST language:

```
PROGRAM UserPrg
VAR
sAppName : STRING;
psAppName : POINTER TO STRING;
sTaskName : STRING;
psTaskName : POINTER TO STRING;
pstTaskInfo : POINTER TO stTaskInfo;
TaskInfo : stTaskInfo;
Info : ERRORCODE;
END_VAR
//INPUTS:
sAppName := 'Application';    //Variable receives the application name.
psAppName := ADR(sAppName); //Pointer with application name.
sTaskName := 'MainTask';    //Variable receives task name.
psTaskName := ADR(sTaskName); //Pointer with task name.
pstTaskInfo := ADR(TaskInfo); //Pointer that receives task info.
//FUNCTION:
//Function call.
Info := GetTaskInfo (psAppName, psTaskName, pstTaskInfo);
//Variable Info receives possible function errors.
```

7.2. Graphic Display

The graphic display available in this product has an important tool for the process control, as through it is possible to recognize possible error conditions, active components or diagnostics presence. Besides, all diagnostics including the I/O modules are presented to the user through the graphic display. For further information regarding the diagnostic key utilization and its visualization see One Touch Diag section.

On figure below, it is possible to observe the available characters in this product graphic display and, next, its respective meanings.

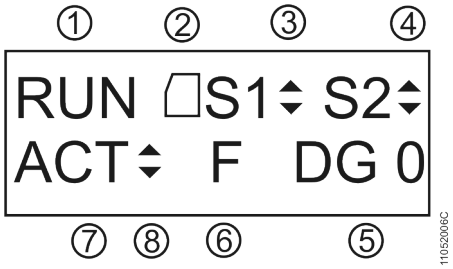


Figure 191: CPU Status Screen

Legend:

- 1. Indication of the CPU status operation. In case the CPU application is running, the state is RUN. In case the CPU application is stopped, the state is STOP and, when is stopped in an application depuration mark, the state is BRKP. For further details, see CPU Operating States section.
- 2. Memory Card presence indication. Further details regarding its installation see Memory Card Installation section.
- 3. COM 1 traffic indication. The up arrow (▲) indicates data transmission and the down arrow (▼) indicates data reception. For further information regarding the COM 1 interface see Serial Interfaces section.
- 4. COM 2 traffic indication. The up arrow (▲) indicates data transmission and the down arrow (▼) indicates data reception. For further information regarding the COM 2 interface see Serial Interfaces section.
- 5. Indication of the CPU active diagnostics quantity. In case the number shown is different than 0 (zero), there are active diagnostics in the CPU. For further details regarding their visualization on the CPU graphic display, through diagnostic key, see One Touch Diag section.
- 6. Forced variables in the CPU indication. In case the "F" character is shown in the graphic display, a variable is being forced by the user, whether symbolic, direct representation or AT. For further information regarding variable forcing see Writing and Forcing Variables section.
- 7. Identification of the CPU redundancy state (message only valid in NX3030 in redundant mode). If the CPU is the active PLC, the ACT information will be presented. The other possible states are NCF (Not-configured), STR (Starting), INA (Inactive) and SBY (Stand-by).
- 8. Indication that the project synchronization is being executed. The up arrow (▲) indicates project data transmission and the down arrow (▼) indicates project data reception. For further information about the project synchronization see Project Synchronization section.

Besides the characters described above, Nexto CPUs can present some messages on the graphic display, correspondent to a process which is being executed at the moment.

The table below present the messages and their respective descriptions:

Message	Description
FORMATTING...	Indicates the CPU is formatting the memory card.
FORMATTING ERROR	Indicates that an error occurred while formatting the memory card by the CPU.
WRONG FORMAT	Indicates that the memory card format is incorrect.
INCORRECT PASSWORD	Indicates the typed password is different from the configured password.
TRANSFERRING...	Indicates the project is being transferred.
TRANSFERRING ERROR	Indicates there is been an error in the project transference caused by some problem in the memory card or its removal during transference.
TRANSFERRING COMPLETE	Indicates the transference has been executed successfully.
TRANSFERRING TIMEOUT	Indicates a time-out has been occurred (communication time expired) during the project transference.
CPU TYPE MISMATCH	Indicates the CPU model is different from the one configured in the project within the memory card.
VERSION MISMATCH	Indicates the CPU version is different from the one configured in the project within the memory card.

Message	Description
APPLICATION CORRUPTED	Indicates the application within the memory card is corrupted.
APPLICATION NOT FOUND	Indicates there is no application in the memory card to be transferred to the CPU.
CRC NOT FOUND	Indicates that the CRC application does not exist.
MCF FILE NOT FOUND	Indicates there is no MCF file in the memory card.
NO TAG	There is no configured tag for the CPU in the MasterTool IEC XE.
NO DESC	There is no configured description for the CPU in the MasterTool IEC XE.
MSG. ERROR	Indicates that there are error(s) on diagnostics message(s) of the requested module(s).
SIGNATURE MISSING	Indicates the product presented an unexpected problem. Get in contact with Altus Technical Support sector.
APP. ERROR RESTARTING	Indicates that occurred an error in the application and the Runtime is restarting the application.
APP. NOT LOADED	Indicates that the runtime will not load the application.
LOADING APP.	Indicates that the runtime will load the application.
WRONG SLOT	Indicates that the CPU is in an incorrect position in the rack.
FATAL ERROR	Indicates that there are serious problems in the CPU startup such as CPU partitions that were not properly mounted. Please, contact Altus customer support.
HW-SW MISMATCH	Indicates that the CPU hardware and software are not compatible because the product presented an unexpected problem. Please, contact Altus customer support.
UPDATING FIRMWARE	Indicates the firmware is being updated in the CPU.
RECEIVING FIRMWARE	Indicates the updating file is being transferred to the CPU.
UPDATED	Shows the firmware version updated in the CPU.
UPDATE ERROR	Indicates an error has occurred during the CPU firmware updating, caused by communication failure or configuration problems.
REBOOTING SYSTEM...	Indicates the CPU is being restarted for the updating to have effect.

Table 243: Other Messages of the Graphic Display

7.3. System Log

The *System Log* is an available feature in the MasterTool IEC XE programmer. It is an important tool for process control, as it makes it possible to find events on CPU that may indicate error conditions, presence of active components or active diagnostics. Such events can be viewed in chronological order with a resolution of milliseconds, with a storage capacity of up to one thousand log entries stored in the CPU internal memory, that can't be removed.

In order to access these Logs, just go to the *Device Tree* and double-click on *Device*, then go to the *Log* tab, where hundreds of operations can be seen, such as: task max cycles, user access, online change, application download and upload, application synchronization between CPUs, firmware update between another events and actions.

In order to view the *Logs*, just need to be connected to a CPU (selected Active Path) and click on . When this button is pressed the Logs are displayed and updated instantly. When the button is not being pressed the Logs will be hold in the screen, it means, these button has two stages, one hold the logs state being updated and in the state the updating is disabled. To no longer show the Logs, press .

It is possible to filter the Logs in 4 different types: warning(s), error(s), exception(s) and information.

Another way to filter the messages displayed to the user is to select the component desired to view.

The Log tab's *Time Stamp* is shown by MasterTool after information provided by the device (CPU). MasterTool can display the Time Stamp in local time (computer's time) or UTC, if *UTC time* checkbox is marked.

ATTENTION

If the device's time or time zone parameter are incorrect, the Time Stamp shown in MasterTool also won't be correct.

For further information about the System Logs please check the MasterTool IEC XE User Manual – MU299609 and the RTC Clock and Time Synchronization subsection of this manual.

ATTENTION

The system logs of the CPUs, starting in firmware version 1.4.0.33 (Nexto) and 1.14.36.0 (Xtorm), are reloaded in the cases of a restart of the CPU or a reboot of the Runtime System, that is, it will be possible to check the older logs when one of these situations occurs.

7.4. Not Loading the Application at Startup

If necessary, the user can choose to not load an existing application on the CPU during its startup. Just power the CPU with the diagnostics button pressed and keep it pressed for until the message "APP. NOT LOADED" is shown in the screen. If a login attempt is made, MasterTool IEC XE software will indicate that there is no application on the CPU. For reloading the application, the CPU must be reset or a new application download must be done.

7.5. Power Supply Failure

The Nexto Series Power Supply (NX8000) has a failure detection system according to the levels defined in its technical features (see Power Supply 30 W 24 Vdc Technical Characteristics - CE114200). There are two ways to diagnose a failure:

1. In case the NX8000 power supply is on with voltage lower than the required minimum limit, a power supply failure diagnostic is generated, which is recognized by the CPU and the message "POWER FAILURE" is shown on the display. When the supply is within the established limits, the CPU recognizes it and automatically is restarted with the user application. The diagnostic will still be active to show to the user that the last initialization suffered a power supply failure.
2. In case the NX8000 has a voltage drop to an inferior value than the minimum required limit and it returns to a higher value within 10 ms, the power supply failure is not recognized by the CPU and the diagnostic is not generated as the system remains intact during this time. But if the voltage drop takes longer than 10 ms, the "POWER FAILURE" message is shown on the CPU screen and the diagnostic is activated.

Figure 192: Power Supply Failure Message

The user can change the value of the variable attributed to the power supply failure to FALSE during the application execution, facilitating the verification and treatment of this diagnostic.

The POWER FAILURE diagnostic is already mapped in a specific memory region, defined as CPU Detailed Diagnostic. This way it is just to use it as global variable. The variable name is described in the detailed diagnostic list in the Diagnostics via Variables section.

7.6. Common Problems

If, at power on the CPU, it does not work, the following items must be verified:

- Is the room temperature within the device supported range?
- Is the rack power supply being fed with the correct voltage?
- Is the power supply module inserted on the far left in the rack (observing the rack by the front view) followed by the Nexto Series CPU?
- Are there network devices, as hubs, switches or routers, powered, interconnected, configured and working properly?

- Is the Ethernet network cable properly connected to the Nexto CPU NET 1 or NET 2 port and to the network device?
- Is the Nexto Series CPU on, in execution mode (Run) and with no diagnostics related to hardware?

If the Nexto CPU indicates the execution mode (Run) but it does not respond to the requested communications, whether through MasterTool IEC XE or protocols, the following items must be verified:

- Is the CPU Ethernet parameters configuration correct?
- Is the respective communication protocol correctly configured in the CPU?
- Are the variables which enable the MODBUS relations properly enabled?

If no problem has been identified, consult the Altus Technical Support.

7.7. Troubleshooting

The table below shows the symptoms of some problems with their possible causes and solutions. If the problem persists, consult the Altus Technical Support.

Symptom	Possible Cause	Solution
Does not power on	Lack of power supply or incorrectly powered.	Verify if the CPU is connected properly in the rack. Power off and take off all modules from the bus, but the power supply and the CPU. Power on the bus and verify the power supply functioning, the external and the one in the rack. Verify if the supply voltage gets to the Nexto power supply contacts and if is correctly polarized.
CPU Screen shows the message WRONG SLOT	CPU in a wrong position.	The CPU must be placed in slots 2 and 3 of rack 0. Put it in the correct slots. CPUs must be placed in slots 2 and 3 of rack 0. Put it in the correct slots.
Does not communicate	Bad contact or bad configuration.	Verify every communication cable connection. Verify the serial and Ethernet interfaces configuration in the MasterTool IEC XE software.
Does not recognize the memory card	Bad connection or not mounted.	Verify if the memory card is properly connected in the compartment. Verify if the memory card was put in the right side, as indicated on the CPU frontal panel. Verify if the memory card wasn't unmounted through MS button, placed on the frontal panel, visualizing the indication on the CPU graphic display.

Table 244: Troubleshooting

7.8. Preventive Maintenance

- It must be verified, each year, if the interconnection cables are connected firmly, without dust accumulation, mainly the protection devices.
- In environments subjected to excessive contamination, the equipment must be periodically cleaned from dust, debris, etc.
- The TVS diodes used for transient protection caused by atmospheric discharges must be periodically inspected, as they might be damaged or destroyed in case the absorbed energy is above limit. In many cases, the failure may not be visual. In critical applications, is recommendable the periodic replacement of the TVS diodes, even if they do not show visual signals of failure.
- Bus tightness and cleanness every six months.
- For further information, see Nexto Series Manual - MU214600.

8. Annex. DNP3 Interoperability

8.1. DNP3 Device Profile

DNP3 **DEVICE PROFILE DOCUMENT**	
Device Identification	
Vendor Name	Altus S/A
Device Name	NX3030
Device Function	Slave
DNP Levels Supported for	Requests: None Responses: None
Connections Supported	IP Networking
Methods to set Configurable Parameters	Software: MasterTool IEC XE
IP Networking	
Type of End Point:	TCP Listening (Outstation Only)
Accepts TCP Connections from	Allows all
IP Address(es) from which TCP Connections are accepted:	*.*.*.*
TCP Listen Port Number	Configurable, range 1 to 65535
TCP Keep-alive timer	Configurable, range 0 to 4294967295
Multiple master connections	Supports up to two masters Based on TCP port number
Time synchronization support	SNTP
Link Layer	
Data Link Address	Configurable, range 0 to 65519
Self Address Support using address 0xFFFC	No
Requires Data Link Layer Confirmation	Never
Maximum number of octets Transmitted in a Data Link Frame	Fixed at 292
Maximum number of octets that can be Received in a Data Link Frame	Fixed at 292
Application Layer	
Maximum number of octets Transmitted in an Application Layer Fragment	Fixed at 2048
Maximum number of octets that can be received in an Application Layer Fragment	Fixed at 2048
Time-out waiting for Application Confirm of solicited response message	Fixed at 10000 ms
Device Trouble Bit IIN1.6	This bit will be set if PLC is not in RUN mode
Event Buffer Overflow Behavior	Discard the oldest event
Sends Multi-Fragment Responses	Yes
Outstation Unsolicited Response Support	
Supports Unsolicited Reporting	No

Table 245: DNP3 Device Profile

8.2. DNP V3.0 Implementation Table

DNP OBJECT GROUP & VARIATION			REQUEST Master may issue Outstation must parse		RESPONSE Master must parse Outstation may issue	
Group Num	**Var Num**	**Description**	**Function Codes (dec)**	**Qualifier Codes (hex)**	**Function Codes (dec)**	**Qualifier Codes (hex)**
1	0	Binary Input – Any Variation	1 (read)	00, 01 (start-stop) 06 (no range, or all)		
1	1	Binary Input – Packed format	1 (read)	00, 01 (start-stop) 06 (no range, or all)	129 (response)	00, 01 (start-stop)
2	0	Binary Input Event – Any Variation	1 (read)	06 (no range, or all) 07, 08 (limited qty)		
2	2	Binary Input Event – With absolute time	1 (read)	06 (no range, or all) 07, 08 (limited qty)	129 (response)	17, 28 (index)
60	1	Class Objects – Class 0 data	1 (read)	06 (no range, or all)		
60	2	Class Objects – Class 1 data	1 (read)	06 (no range, or all) 07, 08 (limited qty)		
80	1	Internal Indications - Packed format	1 (read)	00, 01 (start-stop)	129 (response)	00 (start-stop)
			2 (write)	00 (start-stop) index=7		

Table 246: DNP V3.0 Implementation Table